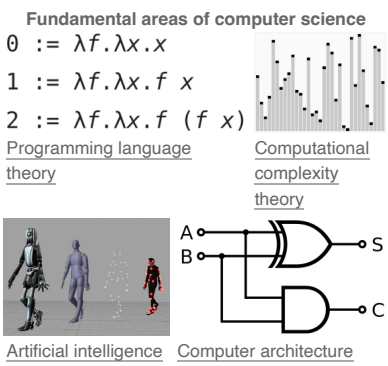
Computer science



Computer science is the study of computation, automation, and information.[1] Computer science spans theoretical disciplines (such as algorithms, theory of computation, information theory, and automation) to practical disciplines (including the design and implementation of hardware and software).[2][3][4] Computer science is generally considered an area of academic research and distinct from computer programming.[5]

Algorithms and data structures are central to computer science.[6] The theory of computation concerns abstract models of computation and general classes of problems that can be solved using them. The fields of cryptography and computer security involve studying the means for secure communication and for preventing security vulnerabilities. Computer graphics and computational geometry address the generation of images. Programming language theory considers different ways to describe computational processes, and database theory concerns the management of repositories of data. Human–computer interaction investigates the interfaces through which humans and computers interact, and software engineering focuses on the design and principles behind developing software. Areas such as operating systems, networks and embedded systems investigate the principles and design behind complex systems. Computer architecture describes the construction of computer components and computer-operated equipment. Artificial intelligence and machine learning aim to synthesize goal-orientated processes such as problem-solving, decision-making, environmental adaptation, planning and learning found in humans and animals. Within artificial intelligence, computer vision aims to understand and process image and video data, while natural language processing aims to understand and process textual and linguistic data.

The fundamental concern of computer science is determining what can and cannot be automated.[7][8][9][10][11] The Turing Award is generally recognized as the highest distinction in computer science.[12][13]

Contents

History

Charles Babbage, sometimes referred to as the "father of computing".[14]

The earliest foundations of what would become computer science predate the invention of the modern digital computer. Machines for calculating fixed numerical tasks such as the abacus have existed since antiquity, aiding in computations such as multiplication and division. Algorithms for performing computations have existed since antiquity, even before the development of sophisticated computing equipment.[16]

Wilhelm Schickard designed and constructed the first working mechanical calculator in 1623.[17] In 1673, Gottfried Leibniz demonstrated a digital mechanical calculator, called the Stepped Reckoner.[18] Leibniz may be considered the first computer scientist and information theorist, because of various reasons, including the fact that he documented the binary number system. In 1820, Thomas de Colmar launched the mechanical calculator industry[note 1] when he invented his simplified arithmometer, the first calculating machine strong enough and reliable enough to be used daily in an office environment. Charles Babbage started the design of the first *automatic mechanical calculator*, his Difference Engine, in 1822, which eventually gave him the idea of the first *programmable mechanical calculator*, his Analytical Engine.[19] He started developing this machine in 1834, and "in less than two years, he had sketched out many of the salient features of the modern computer".[20] "A crucial step was the adoption of a punched card system derived from the Jacquard loom"[20] making it infinitely programmable.[note 2] In 1843, during the translation of a French article on the Analytical Engine, Ada Lovelace wrote, in one of the many notes she included, an algorithm to compute the Bernoulli numbers, which is considered to be the first published algorithm ever specifically tailored for implementation on a computer.[21] Around 1885, Herman Hollerith invented the tabulator, which used punched cards to process statistical information; eventually his company became part of IBM. Following Babbage, although unaware of his earlier work, Percy Ludgate in 1909 published[22] the 2nd of the only two designs for mechanical analytical engines in history. In 1937, one hundred years after Babbage's impossible dream, Howard Aiken convinced IBM, which was making all kinds of punched card equipment and was also in the calculator business[23] to develop his giant programmable calculator, the ASCC/Harvard Mark I, based on Babbage's Analytical Engine, which itself used cards and a central computing unit. When the machine was finished, some hailed it as "Babbage's dream come true".[24]

Ada Lovelace published the first algorithm intended for processing on a computer.[15]

During the 1940s, with the development of new and more powerful computing machines such as the Atanasoff–Berry computer and ENIAC, the term *computer* came to refer to the machines rather than their human predecessors.[25] As it became clear that computers could be used for more than just mathematical calculations, the field of computer science broadened to study computation in general. In 1945, IBM founded the Watson Scientific Computing Laboratory at Columbia University in New York City. The renovated fraternity house on Manhattan's West Side was IBM's first laboratory devoted to pure science. The lab is the forerunner of IBM's Research Division, which today operates research facilities around the world.[26] Ultimately, the close relationship between IBM and Columbia University was instrumental in the emergence of a new scientific discipline, with Columbia offering one of the first academic-credit courses in computer science in 1946.[27] Computer science began to be established as a distinct academic discipline in the 1950s and early 1960s.[28][29] The world's first computer science degree program, the Cambridge Diploma in Computer Science, began at the University of Cambridge Computer Laboratory in 1953. The first computer science department in the United States was formed at Purdue University in 1962.[30] Since practical computers became available, many applications of computing have become distinct areas of study in their own rights.

Etymology

Although first proposed in 1956,[31] the term "computer science" appears in a 1959 article in *Communications of the ACM*,[32] in which Louis Fein argues for the creation of a *Graduate School in Computer Sciences* analogous to the creation of Harvard Business School in 1921.[33] Louis justifies the name by arguing that, like management science, the subject is applied and interdisciplinary in nature, while having the characteristics typical of an academic discipline.[32] His efforts, and those of others such as numerical analyst George Forsythe, were rewarded: universities went on to create such departments, starting with Purdue in 1962.[34] Despite its name, a significant amount of computer science does not involve the study of computers themselves. Because of this, several alternative names have been proposed.[35] Certain departments of major universities prefer the term *computing science*, to emphasize precisely that difference. Danish scientist Peter Naur suggested the term *datalogy*,[36] to reflect the fact that the scientific discipline revolves around data and data treatment, while not necessarily involving computers. The first scientific institution to use the term was the Department of Datalogy at the University of Copenhagen, founded in 1969, with Peter Naur being the first professor in datalogy. The term is used mainly in the Scandinavian countries. An alternative term, also proposed by Naur, is data science; this is now used for a multi-disciplinary field of data analysis, including statistics and databases.

In the early days of computing, a number of terms for the practitioners of the field of computing were suggested in the *Communications of the ACM*—*turingineer*, *turologist*, *flow-charts-man*, *applied meta-mathematician*, and *applied epistemologist*.[37] Three months later in the same journal, *comptologist* was suggested, followed next year by *hypologist*.[38] The term *computics* has also been suggested.[39] In Europe, terms derived from contracted translations of the expression "automatic information" (e.g. "informazione automatica" in Italian) or "information and mathematics" are often used, e.g. *informatique* (French), *Informatik* (German), *informatica* (Italian, Dutch), *informática* (Spanish, Portuguese), *informatika* (Slavic languages and Hungarian) or *pliroforiki* (*πληροφορική*, which means informatics) in Greek. Similar words have also been adopted in the UK (as in the School of Informatics, University of Edinburgh).[40] "In the U.S., however, informatics is linked with applied computing, or computing in the context of another domain."[41]

A folkloric quotation, often attributed to—but almost certainly not first formulated by—Edsger Dijkstra, states that "computer science is no more about computers than astronomy is about telescopes."[note 3] The design and deployment of computers and computer systems is generally considered the province of disciplines other than computer science. For example, the study of computer hardware is usually considered part of computer engineering, while the study of commercial computer systems and their deployment is often called information technology or information systems. However, there has been exchange of ideas between the various computer-related disciplines. Computer science research also often intersects other disciplines, such as cognitive science, linguistics, mathematics, physics, biology, Earth science, statistics, philosophy, and logic.

Computer science is considered by some to have a much closer relationship with mathematics than many scientific disciplines, with some observers saying that computing is a mathematical science.[28] Early computer science was strongly influenced by the work of mathematicians such as Kurt Gödel, Alan Turing, John von Neumann, Rózsa Péter and Alonzo Church and there continues to be a useful interchange of ideas between the two fields in areas such as mathematical logic, category theory, domain theory, and algebra.[31]

The relationship between Computer Science and Software Engineering is a contentious issue, which is further muddied by disputes over what the term "Software Engineering" means, and how computer science is defined.[42] David Parnas, taking a cue from the relationship between other engineering and science disciplines, has claimed that the principal focus of computer science is studying the properties of computation in general, while the principal focus of software engineering is the design of specific computations to achieve practical goals, making the two separate but complementary disciplines.[43]

The academic, political, and funding aspects of computer science tend to depend on whether a department is formed with a mathematical emphasis or with an engineering emphasis. Computer science departments with a mathematics emphasis and with a numerical orientation consider alignment with computational science. Both types of departments tend to make efforts to bridge the field educationally if not across all research.

Philosophy

Epistemology of computer science

Despite the word "science" in its name, there is debate over whether or not computer science is a discipline of science, mathematics, or engineering.[44] Allen Newell and Herbert A. Simon argued in 1975,

> Computer science is an empirical discipline. We would have called it an experimental science, but like astronomy, economics, and geology, some of its unique forms of observation and experience do not fit a narrow stereotype of the experimental method. Nonetheless, they are experiments. Each new machine that is built is an experiment. Actually constructing the machine poses a question to nature; and we listen for the answer by observing the machine in operation and analyzing it by all analytical and measurement means available.[44]

It has since been argued that computer science can be classified as an empirical science since it makes use of empirical testing to evaluate the correctness of programs, but a problem remains in defining the laws and theorems of computer science (if any exist) and defining the nature of experiments in computer science.[44] Proponents of classifying computer science as an engineering discipline argue that the reliability of computational systems is investigated in the same way as bridges in civil engineering and airplanes in aerospace engineering.[44] They also argue that while empirical sciences observe what presently exists, computer science observes what is possible to exist and while scientists discover laws from observation, no proper laws have been found in computer science and it is instead concerned with creating phenomena.[44]

Proponents of classifying computer science as a mathematical discipline argue that computer programs are physical realizations of mathematical entities and programs can be deductively reasoned through mathematical formal methods.[44] Computer scientists Edsger W. Dijkstra and Tony Hoare regard instructions for computer programs as mathematical sentences and interpret formal semantics for programming languages as mathematical axiomatic systems.[44]

Paradigms of computer science

A number of computer scientists have argued for the distinction of three separate paradigms in computer science. Peter Wegner argued that those paradigms are science, technology, and mathematics.[45] Peter Denning's working group argued that they are theory, abstraction (modeling), and design.[46] Amnon H. Eden described them as the "rationalist paradigm" (which treats computer science as a branch of mathematics, which is prevalent in theoretical computer science, and mainly employs deductive reasoning), the "technocratic paradigm" (which might be found in engineering approaches, most prominently in software engineering), and the "scientific paradigm" (which approaches computer-related artifacts from the empirical perspective of natural sciences,[47] identifiable in some branches of artificial intelligence).[48] Computer science focuses on methods involved in design, specification, programming, verification, implementation and testing of human-made computing systems.[49]

Fields

> Computer science is no more about computers than astronomy is about telescopes.
>
> — Edsger Dijkstra

As a discipline, computer science spans a range of topics from theoretical studies of algorithms and the limits of computation to the practical issues of implementing computing systems in hardware and software.[50][51] CSAB, formerly called Computing Sciences Accreditation Board—which is made up of representatives of the Association for Computing Machinery (ACM), and the IEEE Computer Society (IEEE CS)[52]—identifies four areas that it considers crucial to the discipline of computer science: *theory of computation*, *algorithms and data structures*, *programming methodology and languages*, and *computer elements and architecture*. In addition to these four areas, CSAB also identifies fields such as software engineering, artificial intelligence, computer networking and communication, database systems, parallel computation, distributed computation, human–computer interaction, computer graphics, operating systems, and numerical and symbolic computation as being important areas of computer science.[50]

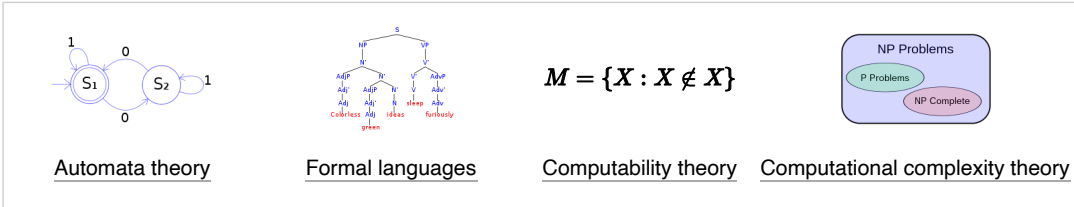
Theoretical computer science

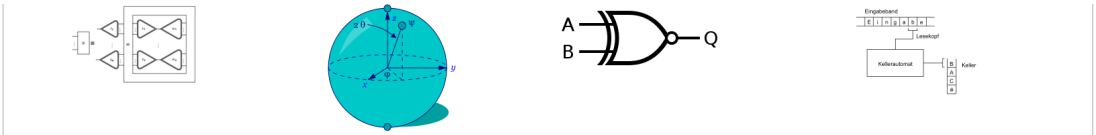
Theoretical Computer Science is mathematical and abstract in spirit, but it derives its motivation from the practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

Theory of computation

According to Peter Denning, the fundamental question underlying computer science is, "What can be automated?"[28] Theory of computation is focused on answering fundamental questions about what can be computed and what amount of resources are required to perform those computations. In an effort to answer the first question, computability theory examines which computational problems are solvable on various theoretical models of computation. The second question is addressed by computational complexity theory, which studies the time and space costs associated with different approaches to solving a multitude of computational problems.

The famous P = NP? problem, one of the Millennium Prize Problems,[53] is an open problem in the theory of computation.

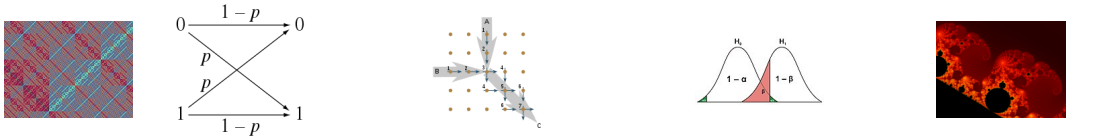




Models of computation | Quantum computing theory | Logic circuit theory | Cellular automata

Information and coding theory

Information theory, closely related to probability and statistics, is related to the quantification of information. This was developed by Claude Shannon to find fundamental limits on signal processing operations such as compressing data and on reliably storing and communicating data.[54] Coding theory is the study of the properties of codes (systems for converting information from one form to another) and their fitness for a specific application. Codes are used for data compression, cryptography, error detection and correction, and more recently also for network coding. Codes are studied for the purpose of designing efficient and reliable data transmission methods. [55]



Coding theory | Channel capacity | Algorithmic information theory | Signal detection theory | Kolmogorov complexity

Data structures and algorithms

Data structures and algorithms are the studies of commonly used computational methods and their computational efficiency.

Analysis of algorithms | Algorithm design | Data structures | Combinatorial optimization | Computational geometry | Randomized algorithms

Programming language theory and formal methods

Programming language theory is a branch of computer science that deals with the design, implementation, analysis, characterization, and classification of programming languages and their individual features. It falls within the discipline of computer science, both depending on and affecting mathematics, software engineering, and linguistics. It is an active research area, with numerous dedicated academic journals.

Formal methods are a particular kind of mathematically based technique for the specification, development and verification of software and hardware systems.[56] The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design. They form an important theoretical underpinning for software engineering, especially where safety or security is involved. Formal methods are a useful adjunct to software testing since they help avoid errors and can also give a framework for testing. For industrial use, tool support is required. However, the high cost of using formal methods means that they are usually only used in the development of high-integrity and life-critical systems, where safety or security is of utmost importance. Formal methods are best described as the application of a fairly broad variety of theoretical computer science fundamentals, in particular logic calculi, formal languages, automata theory, and program semantics, but also type systems and algebraic data types to problems in software and hardware specification and verification.

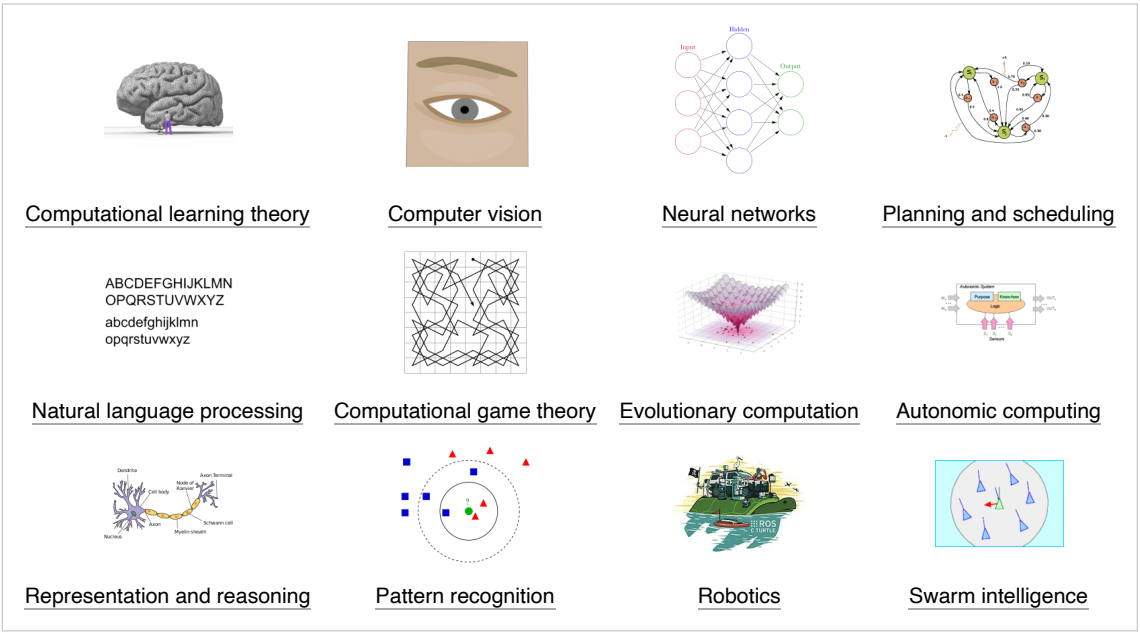
Formal semantics | Type theory | Compiler design | Programming languages | Formal verification | Automated theorem proving

Computer systems and computational processes

Artificial intelligence

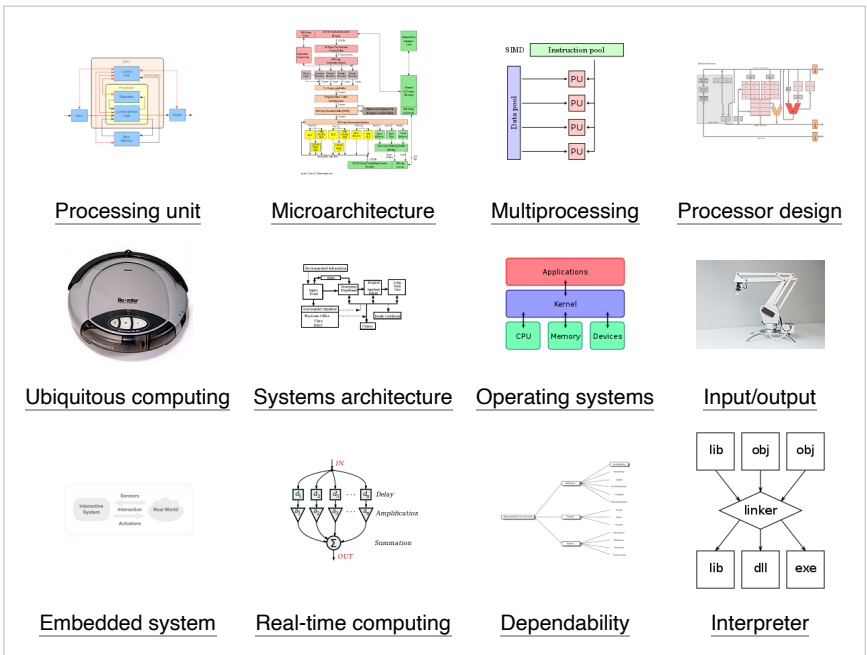
Artificial intelligence (AI) aims to or is required to synthesize goal-orientated processes such as problem-solving, decision-making, environmental adaptation, learning, and communication found in humans and animals. From its origins in cybernetics and in the Dartmouth Conference (1956), artificial intelligence research has been necessarily cross-disciplinary, drawing on areas of expertise such as applied mathematics, symbolic logic, semiotics, electrical engineering, philosophy of mind, neurophysiology, and social intelligence. AI is associated in the popular mind with robotic development, but the main field of practical application has been as an embedded component in areas of software development, which require computational understanding. The starting point in the late 1940s was Alan Turing's question "Can computers think?", and the question remains

effectively unanswered, although the Turing test is still used to assess computer output on the scale of human intelligence. But the automation of evaluative and predictive tasks has been increasingly successful as a substitute for human monitoring and intervention in domains of computer application involving complex real-world data.



Computer architecture and organization

Computer architecture, or digital computer organization, is the conceptual design and fundamental operational structure of a computer system. It focuses largely on the way by which the central processing unit performs internally and accesses addresses in memory.[57] Computer engineers study computational logic and design of computer hardware, from individual processor components, microcontrollers, personal computers to supercomputers and embedded systems. The term "architecture" in computer literature can be traced to the work of Lyle R. Johnson and Frederick P. Brooks, Jr., members of the Machine Organization department in IBM's main research center in 1959.



Concurrent, parallel and distributed computing

Concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other.[58] A number of mathematical models have been developed for general concurrent computation including Petri nets, process calculi and the Parallel Random Access Machine model.[59] When multiple computers are connected in a network while using concurrency, this is known as a distributed system. Computers within that distributed system have their own private memory, and information can be exchanged to achieve common goals.[60]

Computer networks

This branch of computer science aims to manage networks between computers worldwide.

Computer security and cryptography

Computer security is a branch of computer technology with the objective of protecting information from unauthorized access, disruption, or modification while maintaining the accessibility and usability of the system for its intended users.

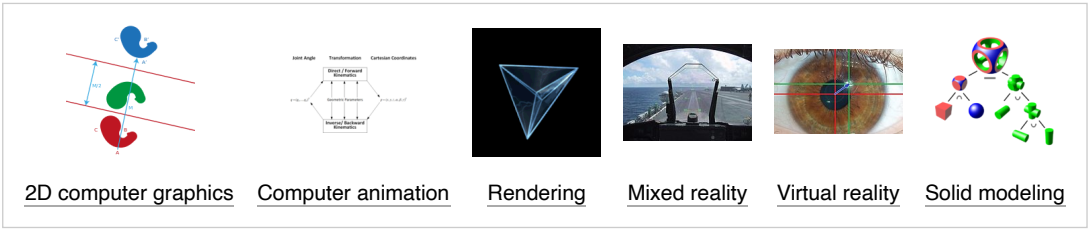
Historical cryptography is the art of writing and deciphering secret messages. Modern cryptography is the scientific study of problems relating to distributed computations that can be attacked.[61] Technologies studied in modern cryptography include symmetric and asymmetric encryption, digital signatures, cryptographic hash functions, key-agreement protocols, blockchain, zero-knowledge proofs, and garbled circuits.

Databases and data mining

A database is intended to organize, store, and retrieve large amounts of data easily. Digital databases are managed using database management systems to store, create, maintain, and search data, through database models and query languages. Data mining is a process of discovering patterns in large data sets.

Computer graphics and visualization

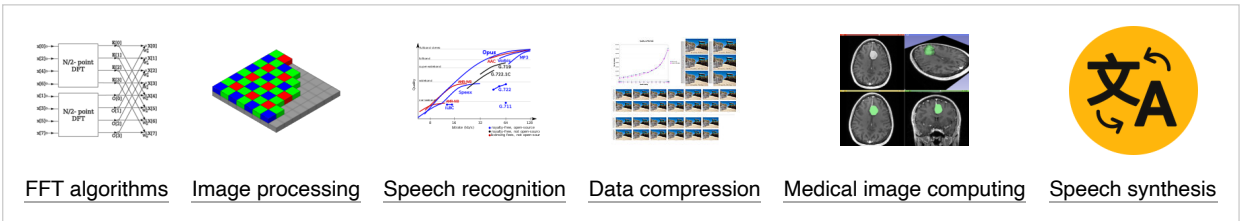
Computer graphics is the study of digital visual contents and involves the synthesis and manipulation of image data. The study is connected to many other fields in computer science, including computer vision, image processing, and computational geometry, and is heavily applied in the fields of special effects and video games.



2D computer graphics | Computer animation | Rendering | Mixed reality | Virtual reality | Solid modeling

Image and sound processing

Information can take the form of images, sound, video or other multimedia. Bits of information can be streamed via signals. Its processing is the central notion of informatics, the European view on computing, which studies information processing algorithms independently of the type of information carrier - whether it is electrical, mechanical or biological. This field plays important role in information theory, telecommunications, information engineering and has applications in medical image computing and speech synthesis, among others. *What is the lower bound on the complexity of fast Fourier transform algorithms?* is one of unsolved problems in theoretical computer science.

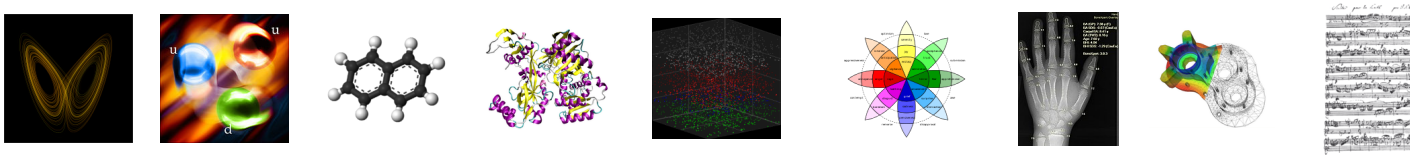


FFT algorithms | Image processing | Speech recognition | Data compression | Medical image computing | Speech synthesis

Applied computer science

Computational science, finance and engineering

Scientific computing (or computational science) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyze and solve scientific problems. A major usage of scientific computing is simulation of various processes, including computational fluid dynamics, physical, electrical, and electronic systems and circuits, as well as societies and social situations (notably war games) along with their habitats, among many others. Modern computers enable optimization of such designs as complete aircraft. Notable in electrical and electronic circuit design are SPICE,[62] as well as software for physical realization of new (or modified) designs. The latter includes essential design software for integrated circuits.[63]



Numerical analysis | Computational physics | Computational chemistry | Bioinformatics | Neuroinformatics | Psychoinformatics | Medical informatics | Computational engineering | Computatic musicolo

Social computing and human–computer interaction

Social computing is an area that is concerned with the intersection of social behavior and computational systems. Human–computer interaction research develops theories, principles, and guidelines for user interface designers.

Software engineering

Software engineering is the study of designing, implementing, and modifying the software in order to ensure it is of high quality, affordable, maintainable, and fast to build. It is a systematic approach to software design, involving the application of engineering practices to software. Software engineering deals with the organizing and analyzing of software—it doesn't just deal with the creation or manufacture of new software, but its internal arrangement and maintenance. For example software testing, systems engineering, technical debt and software development processes.

Discoveries

The philosopher of computing Bill Rapaport noted three *Great Insights of Computer Science*:[64]

- Gottfried Wilhelm Leibniz's, George Boole's, Alan Turing's, Claude Shannon's, and Samuel Morse's insight: there are only *two objects* that a computer has to deal with in order to represent "anything".[note 4]

  > All the information about any computable problem can be represented using only 0 and 1 (or any other bistable pair that can flip-flop between two easily distinguishable states, such as "on/off", "magnetized/de-magnetized", "high-voltage/low-voltage", etc.).

- Alan Turing's insight: there are only *five actions* that a computer has to perform in order to do "anything".

  > Every algorithm can be expressed in a language for a computer consisting of only five basic instructions:[65]
  >
  > - move left one location;
  > - move right one location;
  > - read symbol at current location;
  > - print 0 at current location;
  > - print 1 at current location.

- Corrado Böhm and Giuseppe Jacopini's insight: there are only *three ways of combining* these actions (into more complex ones) that are needed in order for a computer to do "anything".[66]

  > Only three rules are needed to combine any set of basic instructions into more complex ones:
  >
  > - *sequence*: first do this, then do that;
  > - *selection*: IF such-and-such is the case, THEN do this, ELSE do that;
  > - *repetition*: WHILE such-and-such is the case, DO this.
  >
  > Note that the three rules of Boehm's and Jacopini's insight can be further simplified with the use of goto (which means it is more elementary than structured programming).

Programming paradigms

Programming languages can be used to accomplish different tasks in different ways. Common programming paradigms include:

- Functional programming, a style of building the structure and elements of computer programs that treats computation as the evaluation of mathematical functions and avoids state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions or declarations instead of statements.[67]
- Imperative programming, a programming paradigm that uses statements that change a program's state.[68] In much the same way that the imperative mood in natural languages expresses commands, an imperative program consists of commands for the computer to perform. Imperative programming focuses on describing how a program operates.
- Object-oriented programming, a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated. Thus object-oriented computer programs are made out of objects that interact with one another.[69]
- Service-oriented programming, a programming paradigm that uses "services" as the unit of computer work, to design and implement integrated business applications and mission critical software programs

Many languages offer support for multiple paradigms, making the distinction more a matter of style than of technical capabilities.[70]

Research

Conferences are important events for computer science research. During these conferences, researchers from the public and private sectors present their recent work and meet. Unlike in most other academic fields, in computer science, the prestige of conference papers is greater than that of journal publications.[71][72] One proposed explanation for this is the quick development of this relatively new field requires rapid review and distribution of results, a task better handled by conferences than by journals.[73]

Education

Computer Science, known by its near synonyms, **Computing**, **Computer Studies**, has been taught in UK schools since the days of batch processing, mark sensitive cards and paper tape but usually to a select few students.[74] In 1981, the BBC produced a micro-computer and classroom network and Computer Studies became common for GCE O level students (11–16-year-old), and Computer Science to A level students. Its importance was recognised, and it became a compulsory part of the National Curriculum, for Key Stage 3 & 4. In September 2014 it became an entitlement for all pupils over the age of 4.[75]

In the US, with 14,000 school districts deciding the curriculum, provision was fractured.[76] According to a 2010 report by the Association for Computing Machinery (ACM) and Computer Science Teachers Association (CSTA), only 14 out of 50 states have adopted significant education standards for high school computer science.[77] According to a 2021 report, only 51% of high schools in the US offer computer science.[78]

Israel, New Zealand, and South Korea have included computer science in their national secondary education curricula,[79][80] and several others are following.[81]

See also

- Computer engineering
- Computer programming
- Digital Revolution
- Information and communications technology
- Information technology
- List of computer scientists
- List of computer science awards
- List of important publications in computer science

- List of pioneers in computer science
- List of unsolved problems in computer science
- Programming language
- Software engineering

Notes

1. In 1851
2. "The introduction of punched cards into the new engine was important not only as a more convenient form of control than the drums, or because programs could now be of unlimited extent, and could be stored and repeated without the danger of introducing errors in setting the machine by hand; it was important also because it served to crystallize Babbage's feeling that he had invented something really new, something much more than a sophisticated calculating machine." Bruce Collier, 1970
3. See the entry "Computer science" on Wikiquote for the history of this quotation.
4. The word "anything" is written in quotation marks because there are things that computers cannot do. One example is: to answer the question if an arbitrary given computer program will eventually finish or run forever (the Halting problem).

References

1. "What is Computer Science? - Computer Science. The University of York" (https://www.cs.york.ac.uk/undergraduate/what-is-cs/). *www.cs.york.ac.uk*. Retrieved June 11, 2020.
2. "WordNet Search—3.1" (http://wordnetweb.princeton.edu/perl/webwn?s=computer%20scientist). Wordnetweb.princeton.edu. Retrieved May 14, 2012.
3. "Definition of computer science | Dictionary.com" (https://www.dictionary.com/browse/computer-science). *www.dictionary.com*. Retrieved June 11, 2020.
4. "What is Computer Science? | Undergraduate Computer Science at UMD" (https://undergrad.cs.umd.edu/what-computer-science). *undergrad.cs.umd.edu*. Retrieved July 15, 2022.
5. Denning, P.J.; Comer, D.E.; Gries, D.; Mulder, M.C.; Tucker, A.; Turner, A.J.; Young, P.R. (February 1989). "Computing as a discipline" (https://ieeexplore.ieee.org/document/19833). *Computer*. **22** (2): 63–70. doi:10.1109/2.19833 (https://doi.org/10.1109%2F2.19833). ISSN 1558-0814 (https://www.worldcat.org/issn/1558-0814). "Those in the discipline know that computer science encompasses far more than programming."
6. Harel, David (2014). *Algorithmics The Spirit of Computing* (http://worldcat.org/oclc/876384882). Springer Berlin. ISBN 978-3-642-44135-6. OCLC 876384882 (https://www.worldcat.org/oclc/876384882).
7. The MIT Press (April 30, 1980). *What Can Be Automated? Computer Science and Engineering Research Study | The MIT Press* (https://mitpress.mit.edu/books/what-can-be-automated). *mitpress.mit.edu*. Computer Science Series. MIT Press. ISBN 9780262010603.
8. Patton, Richard D.; Patton, Peter C. (2009), Nof, Shimon Y. (ed.), "What Can Be Automated? What Cannot Be Automated?" (https://doi.org/10.1007/978-3-540-78831-7_18), *Springer Handbook of Automation*, Springer Handbooks, Berlin, Heidelberg: Springer, pp. 305–313, doi:10.1007/978-3-540-78831-7_18 (https://doi.org/10.1007%2F978-3-540-78831-7_18), ISBN 978-3-540-78831-7, retrieved March 3, 2022
9. Denning, P.J.; Comer, D.E.; Gries, D.; Mulder, M.C.; Tucker, A.; Turner, A.J.; Young, P.R. (February 1989). "Computing as a discipline" (https://ieeexplore.ieee.org/document/19833). *Computer*. **22** (2): 63–70. doi:10.1109/2.19833 (https://doi.org/10.1109%2F2.19833). ISSN 1558-0814 (https://www.worldcat.org/issn/1558-0814). "The discipline of computing is the systematic study of algorithmic processes that describe and transform information, their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, 'What can be (efficiently) automated?'"
10. Forsythe, George (August 5–10, 1969). "Computer Science and Education". *Proceedings of IFIP Congress 1968*. "The question 'What can be automated?' is one of the most inspiring philosophical and practical questions of contemporary civilization."
11. Knuth, Donald E. (August 1, 1972). "George Forsythe and the development of computer science" (https://doi.org/10.1145/361532.361538). *Communications of the ACM*. **15** (8): 721–726. doi:10.1145/361532.361538 (https://doi.org/10.1145%2F361532.361538). ISSN 0001-0782 (https://www.worldcat.org/issn/0001-0782). S2CID 12512057 (https://api.semanticscholar.org/CorpusID:12512057).
12. Hanson, Vicki L. (January 23, 2017). "Celebrating 50 years of the Turing award" (https://dl.acm.org/doi/fullHtml/10.1145/3033604). *Communications of the ACM*. **60** (2): 5. doi:10.1145/3033604 (https://doi.org/10.1145%2F3033604). ISSN 0001-0782 (https://www.worldcat.org/issn/0001-0782). S2CID 29984960 (https://api.semanticscholar.org/CorpusID:29984960).
13. Scott, Eric; Martins, Marcella Scoczynski Ribeiro; Yafrani, Mohamed El; Volz, Vanessa; Wilson, Dennis G (June 5, 2018). "ACM marks 50 years of the ACM A.M. turing award and computing's greatest achievements" (https://doi.org/10.1145/3231560.3231563). *ACM SIGEVOlution*. **10** (3): 9–11. doi:10.1145/3231560.3231563 (https://doi.org/10.1145%2F3231560.3231563). S2CID 47021559 (https://api.semanticscholar.org/CorpusID:47021559).
14. "Charles Babbage Institute: Who Was Charles Babbage?" (http://www.cbi.umn.edu/about/babbage.html). *cbi.umn.edu*. Retrieved December 28, 2016.
15. "Ada Lovelace | Babbage Engine | Computer History Museum" (http://www.computerhistory.org/babbage/adalovelace/). *www.computerhistory.org*. Retrieved December 28, 2016.
16. "History of Computer Science" (https://cs.uwaterloo.ca/~shallit/Courses/134/history.html#:~:text=In%20the%201960%27s,%20computer%20science,person%20to%20receive%20a%20Ph.). *cs.uwaterloo.ca*. Retrieved July 15, 2022.
17. "Wilhelm Schickard – Ein Computerpionier" (https://web.archive.org/web/20200919014352/https://www.fmi.uni-jena.de/fmimedia/Fakultaet/Institute+und+Abteilungen/Abteilung+f%C3%BCr+Didaktik/GDI/Wilhelm+Schickard.pdf) (PDF) (in German). Archived from the original (http://www.fmi.uni-jena.de/fmimedia/Fakultaet/Institute+und+Abteilungen/Abteilung+f%C3%BCr+Didaktik/GDI/Wilhelm+Schickard.pdf) (PDF) on September 19, 2020. Retrieved December 4, 2016.
18. Keates, Fiona (June 25, 2012). "A Brief History of Computing" (http://blogs.royalsociety.org/history-of-science/2012/06/25/history-of-computing/). *The Repository*. The Royal Society.
19. "Science Museum, Babbage's Analytical Engine, 1834-1871 (Trial model)" (https://collection.sciencemuseumgroup.org.uk/objects/co62245/babbages-analytical-engine-1834-1871-trial-model-analytical-engines). Retrieved May 11, 2020.
20. Anthony Hyman (1982). *Charles Babbage, pioneer of the computer* (https://archive.org/details/charlesbabbagepi0000hyma). ISBN 9780691083032.
21. "A Selection and Adaptation From Ada's Notes found in Ada, The Enchantress of Numbers," by Betty Alexandra Toole Ed.D. Strawberry Press, Mill Valley, CA" (https://web.archive.org/web/20060210172109/http://www.scottlan.edu/lriddle/women/ada-love.htm). Archived from the original (http://www.scottlan.edu/Lriddle/women/ada-love.htm) on February 10, 2006. Retrieved May 4, 2006.
22. "The John Gabriel Byrne Computer Science Collection" (https://web.archive.org/web/20190416071721/https://www.scss.tcd.ie/SCSSTreasuresCatalog/miscellany/TCD-SCSS-X.20121208.002/TCD-SCSS-X.20121208.002.pdf) (PDF). Archived from the original (https://scss.tcd.ie/SCSSTreasuresCatalog/miscellany/TCD-SCSS-X.20121208.002/TCD-SCSS-X.20121208.002.pdf/) on April 16, 2019. Retrieved August 8, 2019.
23. "In this sense Aiken needed IBM, whose technology included the use of punched cards, the accumulation of numerical data, and the transfer of numerical data from one register to another", Bernard Cohen, p.44 (2000)
24. Brian Randell, p. 187, 1975
25. The Association for Computing Machinery (ACM) was founded in 1947.
26. "IBM Archives: 1945" (https://www.ibm.com/ibm/history/history/year_1945.html). Ibm.com. January 23, 2003. Retrieved March 19, 2019.
27. "IBM100 – The Origins of Computer Science" (https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/). Ibm.com. September 15, 1995. Retrieved March 19, 2019.

28. Denning, Peter J. (2000). "Computer Science: The Discipline" (https://web.archive.org/web/20060525195404/http://www.idi.ntnu.no/emner/dif8916/denning.pdf) (PDF). *Encyclopedia of Computer Science*. Archived from the original (http://www.idi.ntnu.no/emner/dif8916/denning.pdf) (PDF) on May 25, 2006.
29. "Some EDSAC statistics" (http://www.cl.cam.ac.uk/conference/EDSAC99/statistics.html). University of Cambridge. Retrieved November 19, 2011.
30. "Computer science pioneer Samuel D. Conte dies at 85" (http://www.cs.purdue.edu/about/conte.html). Purdue Computer Science. July 1, 2002. Retrieved December 12, 2014.
31. Tedre, Matti (2014). *The Science of Computing: Shaping a Discipline*. Taylor and Francis / CRC Press.
32. Louis Fine (1960). "The Role of the University in Computers, Data Processing, and Related Fields". *Communications of the ACM*. **2** (9): 7–14. doi:10.1145/368424.368427 (https://doi.org/10.1145%2F368424.368427). S2CID 6740821 (https://api.semanticscholar.org/CorpusID:6740821).
33. "Stanford University Oral History" (http://library.stanford.edu/guides/stanford-university-oral-history). Stanford University. Retrieved May 30, 2013.
34. Donald Knuth (1972). *"George Forsythe and the Development of Computer Science" (http://www.stanford.edu/dept/ICME/docs/history/forsythe_knuth.pdf). Comms. ACM.* Archived (https://web.archive.org/web/20131020200802/http://www.stanford.edu/dept/ICME/docs/history/forsythe_knuth.pdf) October 20, 2013, at the Wayback Machine
35. Matti Tedre (2006). "The Development of Computer Science: A Sociocultural Perspective" (http://epublications.uef.fi/pub/urn_isbn_952-458-867-6/urn_isbn_952-458-867-6.pdf) (PDF). p. 260. Retrieved December 12, 2014.
36. Peter Naur (1966). "The science of datalogy". *Communications of the ACM*. **9** (7): 485. doi:10.1145/365719.366510 (https://doi.org/10.1145%2F365719.366510). S2CID 47558402 (https://api.semanticscholar.org/CorpusID:47558402).
37. Weiss, E.A.; Corley, Henry P.T. "Letters to the editor". *Communications of the ACM*. **1** (4): 6. doi:10.1145/368796.368802 (https://doi.org/10.1145%2F368796.368802). S2CID 5379449 (https://api.semanticscholar.org/CorpusID:5379449).
38. Communications of the ACM 2(1):p.4
39. IEEE Computer 28(12): p.136
40. P. Mounier-Kuhn, *L'Informatique en France, de la seconde guerre mondiale au Plan Calcul. L'émergence d'une science*, Paris, PUPS, 2010, ch. 3 & 4.
41. Groth, Dennis P. (February 2010). "Why an Informatics Degree?" (http://cacm.acm.org/magazines/2010/2/69363-why-an-informatics-degree). *Communications of the ACM*. Cacm.acm.org.
42. Tedre, M. (2011). "Computing as a Science: A Survey of Competing Viewpoints". *Minds and Machines*. **21** (3): 361–387. doi:10.1007/s11023-011-9240-4 (https://doi.org/10.1007%2Fs11023-011-9240-4). S2CID 14263916 (https://api.semanticscholar.org/CorpusID:14263916).
43. Parnas, D.L. (1998). "Software engineering programmes are not computer science programmes". *Annals of Software Engineering*. **6**: 19–37. doi:10.1023/A:1018949113292 (https://doi.org/10.1023%2FA%3A1018949113292). S2CID 35786237 (https://api.semanticscholar.org/CorpusID:35786237)., p. 19: "Rather than treat software engineering as a subfield of computer science, I treat it as an element of the set, Civil Engineering, Mechanical Engineering, Chemical Engineering, Electrical Engineering, […]"
44. "The Philosophy of Computer Science" (https://plato.stanford.edu/entries/computer-science/#EpisStatCompScie). *The Philosophy of Computer Science (Stanford Encyclopedia of Philosophy)*. Metaphysics Research Lab, Stanford University. 2021.
45. Wegner, P. (October 13–15, 1976). *Research paradigms in computer science—Proceedings of the 2nd international Conference on Software Engineering*. San Francisco, California, United States: IEEE Computer Society Press, Los Alamitos, CA.
46. Denning, P.J.; Comer, D.E.; Gries, D.; Mulder, M.C.; Tucker, A.; Turner, A.J.; Young, P.R. (January 1989). "Computing as a discipline". *Communications of the ACM*. **32**: 9–23. doi:10.1145/63238.63239 (https://doi.org/10.1145%2F63238.63239). S2CID 723103 (https://api.semanticscholar.org/CorpusID:723103).
47. Denning, Peter J. (2007). "Computing is a natural science". *Communications of the ACM*. **50** (7): 13–18. doi:10.1145/1272516.1272529 (https://doi.org/10.1145%2F1272516.1272529). S2CID 20045303 (https://api.semanticscholar.org/CorpusID:20045303).
48. Eden, A.H. (2007). "Three Paradigms of Computer Science" (https://web.archive.org/web/20160215100211/http://www.eden-study.org/articles/2007/three_paradigms_of_computer_science.pdf) (PDF). *Minds and Machines*. **17** (2): 135–167. CiteSeerX 10.1.1.304.7763 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.304.7763). doi:10.1007/s11023-007-9060-8 (https://doi.org/10.1007%2Fs11023-007-9060-8). S2CID 3023076 (https://api.semanticscholar.org/CorpusID:3023076). Archived from the original (http://www.eden-study.org/articles/2007/three_paradigms_of_computer_science.pdf) (PDF) on February 15, 2016.
49. Turner, Raymond; Angius, Nicola (2019). "The Philosophy of Computer Science" (https://plato.stanford.edu/archives/spr2019/entries/computer-science/). In Zalta, Edward N. (ed.). *The Stanford Encyclopedia of Philosophy*.
50. "Computer Science as a Profession" (https://web.archive.org/web/20080617030847/http://www.csab.org/comp_sci_profession.html). Computing Sciences Accreditation Board. May 28, 1997. Archived from the original (http://www.csab.org/comp_sci_profession.html) on June 17, 2008. Retrieved May 23, 2010.
51. Committee on the Fundamentals of Computer Science: Challenges and Opportunities, National Research Council (2004). *Computer Science: Reflections on the Field, Reflections from the Field* (http://www.nap.edu/catalog.php?record_id=11106#toc). National Academies Press. ISBN 978-0-309-09301-9.
52. "CSAB Leading Computer Education" (http://www.csab.org/). CSAB. August 3, 2011. Retrieved November 19, 2011.
53. Clay Mathematics Institute (http://www.claymath.org/millennium/P_vs_NP/) P = NP Archived (https://web.archive.org/web/20131014194456/http://www.claymath.org/millennium/P_vs_NP/) October 14, 2013, at the Wayback Machine
54. P. Collins, Graham (October 14, 2002). "Claude E. Shannon: Founder of Information Theory" (http://www.scientificamerican.com/article.cfm?id=claude-e-shannon-founder). *Scientific American*. Retrieved December 12, 2014.
55. Van-Nam Huynh; Vladik Kreinovich; Songsak Sriboonchitta; 2012. Uncertainty Analysis in Econometrics with Applications. Springer Science & Business Media. p. 63. ISBN 978-3-642-35443-4.
56. Phillip A. Laplante, 2010. Encyclopedia of Software Engineering Three-Volume Set (Print). CRC Press. p. 309. ISBN 978-1-351-24926-3.
57. A. Thisted, Ronald (April 7, 1997). "Computer Architecture" (http://galton.uchicago.edu/~thisted/Distribute/comparch.pdf) (PDF). The University of Chicago.
58. Jiacun Wang, 2017. Real-Time Embedded Systems. Wiley. p. 12. ISBN 978-1-119-42070-5.
59. Gordana Dodig-Crnkovic; Raffaela Giovagnoli; 2013. Computing Nature: Turing Centenary Perspective. Springer Science & Business Media. p. 247. ISBN 978-3-642-37225-4.
60. Simon Elias Bibri; 2018. Smart Sustainable Cities of the Future: The Untapped Potential of Big Data Analytics and Context-Aware Computing for Advancing Sustainability. Springer. p. 74. ISBN 978-3-319-73981-6.
61. Katz, Jonathan (2008). *Introduction to modern cryptography* (https://www.worldcat.org/oclc/137325053). Yehuda Lindell. Boca Raton: Chapman & Hall/CRC. ISBN 978-1-58488-551-1. OCLC 137325053 (https://www.worldcat.org/oclc/137325053).
62. Muhammad H. Rashid, 2016. SPICE for Power Electronics and Electric Power. CRC Press. p. 6. ISBN 978-1-4398-6047-2.
63. "What is an integrated circuit (IC)? A vital component of modern electronics" (https://whatis.techtarget.com/definition/integrated-circuit-IC). *WhatIs.com*. Retrieved November 15, 2021.
64. Rapaport, William J. (September 20, 2013). "What Is Computation?" (http://www.cse.buffalo.edu/~rapaport/computation.html). State University of New York at Buffalo.
65. B. Jack Copeland, 2012. Alan Turing's Electronic Brain: The Struggle to Build the ACE, the World's Fastest Computer. OUP Oxford. p. 107. ISBN 978-0-19-960915-4.
66. Charles W. Herbert, 2010. An Introduction to Programming Using Alice 2.2. Cengage Learning. p. 122. ISBN 0-538-47866-7.
67. Md. Rezaul Karim; Sridhar Alla; 2017. Scala and Spark for Big Data Analytics: Explore the concepts of functional programming, data streaming, and machine learning. Packt Publishing Ltd. p. 87. ISBN 978-1-78355-050-0.
68. Lex Sheehan, 2017. Learning Functional Programming in Go: Change the way you approach your applications using functional programming in Go. Packt Publishing Ltd. p. 16. ISBN 978-1-78728-604-7.
69. Evelio Padilla, 2015. Substation Automation Systems: Design and Implementation. Wiley. p. 245. ISBN 978-1-118-98730-8.

70. "Multi-Paradigm Programming Language" (https://web.archive.org/web/20130821052407/https://developer.mozilla.org/en-US/docs/multiparadigmlanguage.html). *developer.mozilla.org*. Mozilla Foundation. Archived from the original (https://developer.mozilla.org/en-US/docs/multiparadigmlanguage.html) on August 21, 2013.
71. Meyer, Bertrand (April 2009). "Viewpoint: Research evaluation for computer science". *Communications of the ACM*. **25** (4): 31–34. doi:10.1145/1498765.1498780 (https://doi.org/10.1145%2F1498765.1498780). S2CID 8625066 (https://api.semanticscholar.org/CorpusID:8625066).
72. Patterson, David (August 1999). "Evaluating Computer Scientists and Engineers For Promotion and Tenure" (http://cra.org/resources/bp-view/evaluating_computer_scientists_and_engineers_for_promotion_and_tenure/). Computing Research Association.
73. Fortnow, Lance (August 2009). "Viewpoint: Time for Computer Science to Grow Up" (http://cacm.acm.org/magazines/2009/8/34492-viewpoint-time-for-computer-science-to-grow-up/fulltext). *Communications of the ACM*. **52** (8): 33–35. doi:10.1145/1536616.1536631 (https://doi.org/10.1145%2F1536616.1536631).
74. Burns, Judith (April 3, 2016). "Computer science A-level 1970s style" (https://www.bbc.co.uk/news/education-35890450). Retrieved February 9, 2019.
75. Jones, Michael (October 1915). "Developing a Computer Science Curriculum in England: Exploring Approaches in the USA" (https://web.archive.org/web/20161022182632/https://www.wcmt.org.uk/sites/default/files/report-documents/Jones%20M%20Report%202015%20%20Final.pdf) (PDF). Winston Churchill Memorial Trust. Archived from the original (https://www.wcmt.org.uk/sites/default/files/report-documents/Jones%20M%20Report%202015%20%20Final.pdf) (PDF) on October 22, 2016. Retrieved February 9, 2019.
76. "Computer Science: Not Just an Elective Anymore" (http://www.edweek.org/ew/articles/2014/02/26/22computer_ep.h33.html). *Education Week*. February 25, 2014.
77. Wilson, Cameron; Sudol, Leigh Ann; Stephenson, Chris; Stehlik, Mark (2010). "Running on Empty: The Failure to Teach K–12 Computer Science in the Digital Age" (http://runningonempty.acm.org/fullreport2.pdf) (PDF). ACM.
78. "2021 State of computer science education: Accelerating action through advocacy" (https://advocacy.code.org/2021_state_of_cs.pdf) (PDF). Code.org, CSTA, & ECEP Alliance. 2021.
79. "A is for algorithm" (https://www.economist.com/news/international/21601250-global-push-more-computer-science-classrooms-starting-bear-fruit). *The Economist*. April 26, 2014.
80. "Computing at School International comparisons" (https://web.archive.org/web/20130508214038/http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf) (PDF). Archived from the original (http://www.computingatschool.org.uk/data/uploads/internationalcomparisons-v5.pdf) (PDF) on May 8, 2013. Retrieved July 20, 2015.
81. "Adding Coding to the Curriculum" (https://ghostarchive.org/archive/20220101/https://www.nytimes.com/2014/03/24/world/europe/adding-coding-to-the-curriculum.html). *The New York Times*. March 23, 2014. Archived from the original (https://www.nytimes.com/2014/03/24/world/europe/adding-coding-to-the-curriculum.html) on January 1, 2022.

Further reading

Overview

- Tucker, Allen B. (2004). *Computer Science Handbook* (2nd ed.). Chapman and Hall/CRC. ISBN 978-1-58488-360-9.
  - "Within more than 70 chapters, every one new or significantly revised, one can find any kind of information and references about computer science one can imagine. […] all in all, there is absolute nothing about Computer Science that can not be found in the 2.5 kilogram-encyclopaedia with its 110 survey articles […]." (Christoph Meinel, *Zentralblatt MATH*)
- van Leeuwen, Jan (1994). *Handbook of Theoretical Computer Science*. The MIT Press. ISBN 978-0-262-72020-5.
  - "[…] this set is the most unique and possibly the most useful to the [theoretical computer science] community, in support both of teaching and research […]. The books can be used by anyone wanting simply to gain an understanding of one of these areas, or by someone desiring to be in research in a topic, or by instructors wishing to find timely information on a subject they are teaching outside their major areas of expertise." (Rocky Ross, *SIGACT News*)
- Ralston, Anthony; Reilly, Edwin D.; Hemmendinger, David (2000). *Encyclopedia of Computer Science* (http://portal.acm.org/ralston.cfm) (4th ed.). Grove's Dictionaries. ISBN 978-1-56159-248-7.
  - "Since 1976, this has been the definitive reference work on computer, computing, and computer science. […] Alphabetically arranged and classified into broad subject areas, the entries cover hardware, computer systems, information and data, software, the mathematics of computing, theory of computation, methodologies, applications, and computing milieu. The editors have done a commendable job of blending historical perspective and practical reference information. The encyclopedia remains essential for most public and academic library reference collections." (Joe Accardin, Northeastern Illinois Univ., Chicago)
- Edwin D. Reilly (2003). *Milestones in Computer Science and Information Technology* (https://archive.org/details/milestonesincomp0000reil). Greenwood Publishing Group. ISBN 978-1-57356-521-9.

Selected literature

- Knuth, Donald E. (1996). *Selected Papers on Computer Science*. CSLI Publications, Cambridge University Press.
- Collier, Bruce (1990). *The little engine that could've: The calculating machines of Charles Babbage* (http://robroy.dyndns.info/collier/index.html). Garland Publishing Inc. ISBN 978-0-8240-0043-1.
- Cohen, Bernard (2000). *Howard Aiken, Portrait of a computer pioneer*. The MIT press. ISBN 978-0-262-53179-5.
- Tedre, Matti (2014). *The Science of Computing: Shaping a Discipline*. CRC Press, Taylor & Francis.
- Randell, Brian (1973). *The origins of Digital computers, Selected Papers*. Springer-Verlag. ISBN 978-3-540-06169-4.
  - "Covering a period from 1966 to 1993, its interest lies not only in the content of each of these papers – still timely today – but also in their being put together so that ideas expressed at different times complement each other nicely." (N. Bernard, *Zentralblatt MATH*)

Articles

- Peter J. Denning. *Is computer science science? (http://portal.acm.org/citation.cfm?id=1053309&coll=&dl=ACM&CFID=15151515&CFTOKEN=618618)*, Communications of the ACM, April 2005.
- Peter J. Denning, *Great principles in computing curricula (http://portal.acm.org/citation.cfm?id=971303&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618)*, Technical Symposium on Computer Science Education, 2004.
- Research evaluation for computer science, Informatics Europe report (http://www.eqanie.eu/media/Como%20Conference/Tanca-Research_Assessment_A_new_Initiative_by_Informatics_Europe.pdf) Archived (https://web.archive.org/web/20171018181136/http://www.eqanie.eu/media/Com

o%20Conference/Tanca-Research_Assessment_A_new_Initiative_by_Informatics_Europe.pdf) October 18, 2017, at the Wayback Machine. Shorter journal version: Bertrand Meyer, Christine Choppy, Jan van Leeuwen and Jorgen Staunstrup, *Research evaluation for computer science*, in Communications of the ACM, vol. 52, no. 4, pp. 31–34, April 2009.

Curriculum and classification

- Association for Computing Machinery. 1998 ACM Computing Classification System (https://web.archive.org/web/20080828002940/http://www.acm.org/class/1998/overview.html). 1998.
- Joint Task Force of Association for Computing Machinery (ACM), Association for Information Systems (AIS) and IEEE Computer Society (IEEE CS). Computing Curricula 2005: The Overview Report (https://web.archive.org/web/20141021153204/http://www.acm.org/education/curric_vols/CC2005-March06Final.pdf). September 30, 2005.
- Norman Gibbs, Allen Tucker. "A model curriculum for a liberal arts degree in computer science". *Communications of the ACM*, Volume 29 Issue 3, March 1986.

External links

- Computer science (https://curlie.org/Computers/Computer_Science/) at Curlie
- Scholarly Societies in Computer Science (http://www.lib.uwaterloo.ca/society/compsci_soc.html) Archived (https://web.archive.org/web/20110623002546/http://www.lib.uwaterloo.ca/society/compsci_soc.html) June 23, 2011, at the Wayback Machine
- What is Computer Science? (https://www.youtube.com/watch?v=fjMU-km-Cso)
- Best Papers Awards in Computer Science since 1996 (http://jeffhuang.com/best_paper_awards.html)
- Photographs of computer scientists (http://se.ethz.ch/~meyer/gallery/) by Bertrand Meyer
- EECS.berkeley.edu (http://www.eecs.berkeley.edu/department/history.shtml)

Bibliography and academic search engines

- CiteSeer^x (http://citeseerx.ist.psu.edu/) (article): search engine, digital library and repository for scientific and academic papers with a focus on computer and information science.
- DBLP Computer Science Bibliography (http://dblp.uni-trier.de/) (article): computer science bibliography website hosted at Universität Trier, in Germany.
- The Collection of Computer Science Bibliographies (http://liinwww.ira.uka.de/bibliography/) (Collection of Computer Science Bibliographies)

Professional organizations

- Association for Computing Machinery (http://www.acm.org/)
- IEEE Computer Society (http://www.computer.org/)
- Informatics Europe (http://www.informatics-europe.org/)
- AAAI (http://www.aaai.org/home.html)
- AAAS Computer Science (https://web.archive.org/web/20160205000119/http://membercentral.aaas.org/categories/computer-science)

Misc

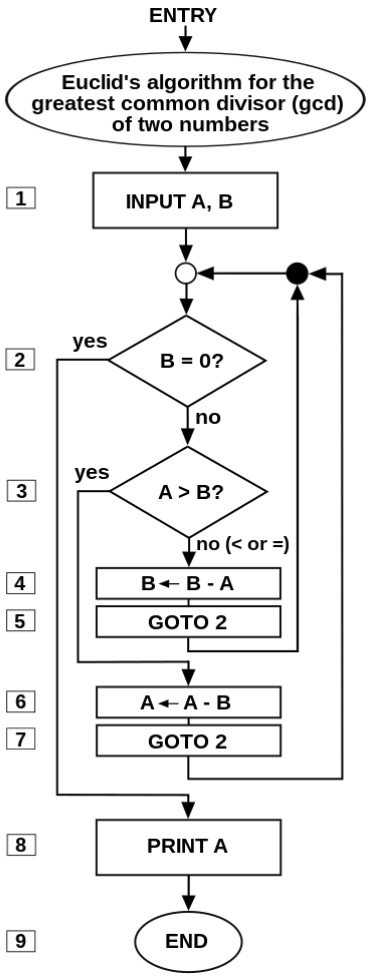
- Computer Science—Stack Exchange (https://cs.stackexchange.com/): a community-run question-and-answer site for computer science
- What is computer science (http://www.cs.bu.edu/AboutCS/WhatIsCS.pdf) Archived (https://web.archive.org/web/20150218130340/http://www.cs.bu.edu/AboutCS/WhatIsCS.pdf) February 18, 2015, at the Wayback Machine
- Is computer science science? (https://web.archive.org/web/20170810205524/https://www.cs.mtu.edu/~john/jenning.pdf)
- Computer Science (Software) Must be Considered as an Independent Discipline. (https://www.researchgate.net/publication/306078165_Computer_Science_Software_Must_be_Considered_as_an_Independent_Discipline_Computer_Science_Software_must_not_be_Treated_as_a_Sub-Domain_or_Subset_of_Mathematics)

Algorithm

In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ (listen)) is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation.[1] Algorithms are used as specifications for performing calculations and data processing. More advanced algorithms can perform automated deductions (referred to as automated reasoning) and use mathematical and logical tests to divert the code execution through various routes (referred to as automated decision-making). Using human characteristics as descriptors of machines in metaphorical ways was already practiced by Alan Turing with terms such as "memory", "search" and "stimulus".[2]

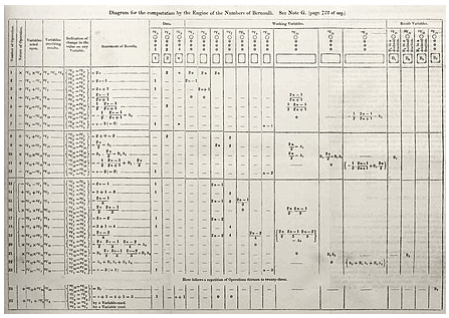
In contrast, a heuristic is an approach to problem solving that may not be fully specified or may not guarantee correct or optimal results, especially in problem domains where there is no well-defined correct or optimal result.[3]

As an effective method, an algorithm can be expressed within a finite amount of space and time,[4] and in a well-defined formal language[5] for calculating a function.[6] Starting from an initial state and initial input (perhaps empty),[7] the instructions describe a computation that, when executed, proceeds through a finite[8] number of well-defined successive states, eventually producing "output"[9] and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.[10]



Flowchart of an algorithm (Euclid's algorithm) for calculating the greatest common divisor (g.c.d.) of two numbers *a* and *b* in locations named A and B. The algorithm proceeds by successive subtractions in two loops: IF the test B ≥ A yields "yes" or "true" (more accurately, the *number b* in location B is greater than or equal to the *number a* in location A) THEN, the algorithm specifies B ← B – A (meaning the number *b* – *a* replaces the old *b*). Similarly, IF A > B, THEN A ← A – B. The process terminates when (the contents of) B is 0, yielding the g.c.d. in A. (Algorithm derived from Scott 2009:13; symbols and drawing style from Tausworthe 1977).

Contents



Ada Lovelace's diagram from "note G", the first published computer algorithm

History

The concept of algorithm has existed since antiquity. Arithmetic algorithms, such as a division algorithm, were used by ancient Babylonian mathematicians c. 2500 BC and Egyptian mathematicians c. 1550 BC.[11] Greek mathematicians later used algorithms in 240 BC in the sieve of Eratosthenes for finding prime numbers, and the Euclidean algorithm for finding the greatest common divisor of two numbers.[12] Arabic mathematicians such as al-Kindi in the 9th century used cryptographic algorithms for code-breaking, based on frequency analysis.[13]

The word *algorithm* is derived from the name of the 9th-century Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī, whose nisba (identifying him as from Khwarazm) was Latinized *as Algoritmi (Arabized Persian* الخوارزمی *c. 780–850).*[14][15] Muḥammad ibn Mūsā al-Khwārizmī was a mathematician, astronomer, geographer, and scholar in the House of Wisdom in Baghdad, whose name means 'the native of Khwarazm', a region that was part of Greater Iran and is now in Uzbekistan.[16][17] About 825, al-Khwarizmi wrote an Arabic language treatise on the Hindu–Arabic numeral system, which was translated into Latin during the 12th century. The

manuscript starts with the phrase *Dixit Algorizmi* ('Thus spake Al-Khwarizmi'), where "Algorizmi" was the translator's Latinization of Al-Khwarizmi's name.[18] Al-Khwarizmi was the most widely read mathematician in Europe in the late Middle Ages, primarily through another of his books, the Algebra.[19] In late medieval Latin, *algorismus*, English 'algorism', the corruption of his name, simply meant the "decimal number system".[20] In the 15th century, under the influence of the Greek word ἀριθμός (*arithmos*), 'number' (*cf.* 'arithmetic'), the Latin word was altered to *algorithmus*, and the corresponding English term 'algorithm' is first attested in the 17th century; the modern sense was introduced in the 19th century.[21]

Indian mathematics was predominantly algorithmic. Algorithms that are representative of the Indian mathematical tradition range from the ancient *Śulbasūtrās* to the medieval texts of the Kerala School.[22]

In English, the word *algorithm* was first used in about 1230 and then by Chaucer in 1391. English adopted the French term, but it was not until the late 19th century that "algorithm" took on the meaning that it has in modern English.[23]

Another early use of the word is from 1240, in a manual titled *Carmen de Algorismo* composed by Alexandre de Villedieu. It begins with:

> *Haec algorismus ars praesens dicitur, in qua / Talibus Indorum fruimur bis quinque figuris.*

which translates to:

> Algorism is the art by which at present we use those Indian figures, which number two times five.

The poem is a few hundred lines long and summarizes the art of calculating with the new styled Indian dice (*Tali Indorum*), or Hindu numerals.[24]

A partial formalization of the modern concept of algorithm began with attempts to solve the *Entscheidungsproblem* (decision problem) posed by David Hilbert in 1928. Later formalizations were framed as attempts to define "effective calculability"[25] or "effective method".[26] Those formalizations included the Gödel–Herbrand–Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines of 1936–37 and 1939.

Informal definition

An informal definition could be "a set of rules that precisely defines a sequence of operations",[27] which would include all computer programs (including programs that do not perform numeric calculations), and (for example) any prescribed bureaucratic procedure[28] or cook-book recipe.[29]

In general, a program is only an algorithm if it stops eventually[30]—even though infinite loops may sometimes prove desirable.

A prototypical example of an algorithm is the Euclidean algorithm, which is used to determine the maximum common divisor of two integers; an example (there are others) is described by the flowchart above and as an example in a later section.

Boolos, Jeffrey & 1974, 1999 offer an informal meaning of the word "algorithm" in the following quotation:

> No human being can write fast enough, or long enough, or small enough† (†"smaller and smaller without limit ... you'd be trying to write on molecules, on atoms, on electrons") to list all members of an enumerably infinite set by writing out their names, one after another, in some notation. But humans can do something equally useful, in the case of certain enumerably infinite sets: They can give *explicit instructions for determining the **n**th member of the set*, for arbitrary finite *n*. Such instructions are to be given quite explicitly, in a form in which *they could be followed by a computing machine*, or by a *human who is capable of carrying out only very elementary operations on symbols*.[31]

An "enumerably infinite set" is one whose elements can be put into one-to-one correspondence with the integers. Thus Boolos and Jeffrey are saying that an algorithm implies instructions for a process that "creates" output integers from an *arbitrary* "input" integer or integers that, in theory, can be arbitrarily large. For example, an algorithm can be an algebraic equation such as $y = m + n$ (i.e., two arbitrary "input variables" m and n that produce an output y), but various authors' attempts to define the notion indicate that the word implies much more than this, something on the order of (for the addition example):

> Precise instructions (in a language understood by "the computer")[32] for a fast, efficient, "good"[33] process that specifies the "moves" of "the computer" (machine or human, equipped with the necessary internally contained information and capabilities)[34] to find, decode, and then process arbitrary input integers/symbols *m* and *n*, symbols *+* and *=* ... and "effectively"[35] produce, in a "reasonable" time,[36] output-integer *y* at a specified place and in a specified format.

The concept of *algorithm* is also used to define the notion of decidability—a notion that is central for explaining how formal systems come into being starting from a small set of axioms and rules. In logic, the time that an algorithm requires to complete cannot be measured, as it is not apparently related to the customary physical dimension. From such uncertainties, that characterize ongoing work, stems the unavailability of a definition of *algorithm* that suits both concrete (in some sense) and abstract usage of the term.

Most algorithms are intended to be implemented as computer programs. However, algorithms are also implemented by other means, such as in a biological neural network (for example, the human brain implementing arithmetic or an insect looking for food), in an electrical circuit, or in a mechanical device.

Formalization

Algorithms are essential to the way computers process data. Many computer programs contain algorithms that detail the specific instructions a computer should perform—in a specific order—to carry out a specified task, such as calculating employees' paychecks or printing students' report cards. Thus, an algorithm can be considered to be any sequence of operations that can be simulated by a Turing-complete system. Authors who assert this thesis include Minsky (1967), Savage (1987) and Gurevich (2000):

> Minsky: "But we will also maintain, with Turing ... that any procedure which could "naturally" be called effective, can, in fact, be realized by a (simple) machine. Although this may seem extreme, the arguments ... in its favor are hard to refute".[37] Gurevich:

> "... Turing's informal argument in favor of his thesis justifies a stronger thesis: every algorithm can be simulated by a Turing machine ... according to Savage [1987], an algorithm is a computational process defined by a Turing machine".[38]

Turing machines can define computational processes that do not terminate. The informal definitions of algorithms generally require that the algorithm always terminates. This requirement renders the task of deciding whether a formal procedure is an algorithm impossible in the general case—due to a major theorem of computability theory known as the halting problem.

Typically, when an algorithm is associated with processing information, data can be read from an input source, written to an output device and stored for further processing. Stored data are regarded as part of the internal state of the entity performing the algorithm. In practice, the state is stored in one or more data structures.

For some of these computational processes, the algorithm must be rigorously defined: specified in the way it applies in all possible circumstances that could arise. This means that any conditional steps must be systematically dealt with, case-by-case; the criteria for each case must be clear (and computable).

Because an algorithm is a precise list of precise steps, the order of computation is always crucial to the functioning of the algorithm. Instructions are usually assumed to be listed explicitly, and are described as starting "from the top" and going "down to the bottom"—an idea that is described more formally by *flow of control*.

So far, the discussion on the formalization of an algorithm has assumed the premises of imperative programming. This is the most common conception—one which attempts to describe a task in discrete, "mechanical" means. Unique to this conception of formalized algorithms is the assignment operation, which sets the value of a variable. It derives from the intuition of "memory" as a scratchpad. An example of such an assignment can be found below.

For some alternate conceptions of what constitutes an algorithm, see functional programming and logic programming.

Expressing algorithms

Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, drakon-charts, programming languages or control tables (processed by interpreters). Natural language expressions of algorithms tend to be verbose and ambiguous, and are rarely used for complex or technical algorithms. Pseudocode, flowcharts, drakon-charts and control tables are structured ways to express algorithms that avoid many of the ambiguities common in the statements based on natural language. Programming languages are primarily intended for expressing algorithms in a form that can be executed by a computer, but are also often used as a way to define or document algorithms.

There is a wide variety of representations possible and one can express a given Turing machine program as a sequence of machine tables (see finite-state machine, state transition table and control table for more), as flowcharts and drakon-charts (see state diagram for more), or as a form of rudimentary machine code or assembly code called "sets of quadruples" (see Turing machine for more).

Representations of algorithms can be classed into three accepted levels of Turing machine description, as follows:[39]

1 High-level description

"...prose to describe an algorithm, ignoring the implementation details. At this level, we do not need to mention how the machine manages its tape or head."

2 Implementation description

"...prose used to define the way the Turing machine uses its head and the way that it stores data on its tape. At this level, we do not give details of states or transition function."

3 Formal description

Most detailed, "lowest level", gives the Turing machine's "state table".

For an example of the simple algorithm "Add m+n" described in all three levels, see Examples.

Design

Algorithm design refers to a method or a mathematical process for problem-solving and engineering algorithms. The design of algorithms is part of many solution theories of operation research, such as dynamic programming and divide-and-conquer. Techniques for designing and implementing algorithm designs are also called algorithm design patterns,[40] with examples including the template method pattern and the decorator pattern.

One of the most important aspects of algorithm design is resource (run-time, memory usage) efficiency; the big O notation is used to describe e.g. an algorithm's run-time growth as the size of its input increases.

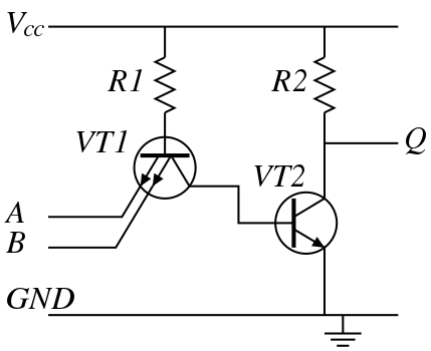
Typical steps in the development of algorithms:

1. Problem definition
2. Development of a model
3. Specification of the algorithm
4. Designing an algorithm
5. Checking the correctness of the algorithm
6. Analysis of algorithm
7. Implementation of algorithm
8. Program testing
9. Documentation preparation

Computer algorithms

"Elegant" (compact) programs, "good" (fast) programs : The notion of "simplicity and elegance" appears informally in Knuth and precisely in Chaitin:

> Knuth: " ... we want *good* algorithms in some loosely defined aesthetic sense. One criterion ... is the length of time taken to perform the algorithm .... Other criteria are adaptability of the algorithm to computers, its simplicity and elegance, etc."[41]
>
> Chaitin: " ... a program is 'elegant,' by which I mean that it's the smallest possible program for producing the output that it does"[42]



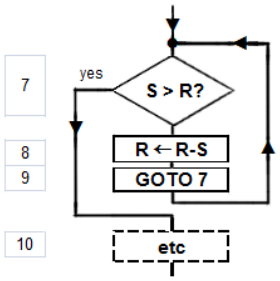
Logical NAND algorithm implemented electronically in 7400 chip

Chaitin prefaces his definition with: "I'll show you can't prove that a program is 'elegant' "—such a proof would solve the Halting problem (ibid).

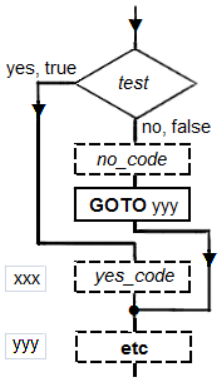
Algorithm versus function computable by an algorithm: For a given function multiple algorithms may exist. This is true, even without expanding the available instruction set available to the programmer. Rogers observes that "It is ... important to distinguish between the notion of *algorithm*, i.e. procedure and the notion of *function computable by algorithm*, i.e. mapping yielded by procedure. The same function may have several different algorithms".[43]

Unfortunately, there may be a tradeoff between goodness (speed) and elegance (compactness)—an elegant program may take more steps to complete a computation than one less elegant. An example that uses Euclid's algorithm appears below.

Computers (and computors), models of computation: A computer (or human "computor"[44]) is a restricted type of machine, a "discrete deterministic mechanical device"[45] that blindly follows its instructions.[46] Melzak's and Lambek's primitive models[47] reduced this notion to four elements: (i) discrete, distinguishable *locations*, (ii) discrete, indistinguishable *counters*[48] (iii) an agent, and (iv) a list of instructions that are *effective* relative to the capability of the agent.[49]



WHILE S ≤ R **DO** R ← R - S



IF test=true **THEN** *yes_code* **ELSE** no_code

Flowchart examples of the canonical Böhm-Jacopini structures: the SEQUENCE (rectangles descending the page), the WHILE-DO and the IF-THEN-ELSE. The three structures are made of the primitive conditional GOTO (`IF test THEN GOTO step xxx`, shown as diamond), the unconditional GOTO (rectangle), various assignment operators (rectangle), and HALT (rectangle). Nesting of these structures inside assignment-blocks result in complex diagrams (cf. Tausworthe 1977:100, 114).

Minsky describes a more congenial variation of Lambek's "abacus" model in his "Very Simple Bases for Computability".[50] Minsky's machine proceeds sequentially through its five (or six, depending on how one counts) instructions unless either a conditional IF-THEN GOTO or an unconditional GOTO changes program flow out of sequence. Besides HALT, Minsky's machine includes three *assignment* (replacement, substitution)[51] operations: ZERO (e.g. the contents of location replaced by 0: L ← 0), SUCCESSOR (e.g. L ← L+1), and DECREMENT (e.g. L ← L − 1).[52] Rarely must a programmer write "code" with such a limited instruction set. But Minsky shows (as do Melzak and Lambek) that his machine is Turing complete with only four general *types* of instructions: conditional GOTO, unconditional GOTO, assignment/replacement/substitution, and HALT. However, a few different assignment instructions (e.g. DECREMENT, INCREMENT, and ZERO/CLEAR/EMPTY for a Minsky machine) are also required for Turing-completeness; their exact specification is somewhat up to the designer. The unconditional GOTO is a convenience; it can be constructed by initializing a dedicated location to zero e.g. the instruction " Z ← 0 "; thereafter the instruction IF Z=0 THEN GOTO xxx is unconditional.

Simulation of an algorithm: computer (computor) language: Knuth advises the reader that "the best way to learn an algorithm is to try it . . . immediately take pen and paper and work through an example".[53] But what about a simulation or execution of the real thing? The programmer must translate the algorithm into a language that the simulator/computer/computor can *effectively* execute. Stone gives an example of this: when computing the roots of a quadratic equation the computor must know how to take a square root. If they don't, then the algorithm, to be effective, must provide a set of rules for extracting a square root.[54]

This means that the programmer must know a "language" that is effective relative to the target computing agent (computer/computor).

But what model should be used for the simulation? Van Emde Boas observes "even if we base complexity theory on abstract instead of concrete machines, arbitrariness of the choice of a model remains. It is at this point that the notion of *simulation* enters".[55] When speed is being measured, the instruction set matters. For example, the subprogram in Euclid's algorithm to compute the remainder would execute much faster if the programmer had a "modulus" instruction available rather than just subtraction (or worse: just Minsky's "decrement").

Structured programming, canonical structures: Per the Church–Turing thesis, any algorithm can be computed by a model known to be Turing complete, and per Minsky's demonstrations, Turing completeness requires only four instruction types—conditional GOTO, unconditional GOTO, assignment, HALT. Kemeny and Kurtz observe that, while "undisciplined" use of unconditional GOTOs and conditional IF-THEN GOTOs can result in "spaghetti code", a programmer can write structured programs using only these instructions; on the other hand "it is also possible, and not too hard, to write badly structured programs in a structured language".[56] Tausworthe augments the three Böhm-Jacopini canonical structures:[57] SEQUENCE, IF-THEN-ELSE, and WHILE-DO, with two more: DO-WHILE and CASE.[58] An additional benefit of a structured program is that it lends itself to proofs of correctness using mathematical induction.[59]

Canonical flowchart symbols[60]: The graphical aide called a flowchart, offers a way to describe and document an algorithm (and a computer program of one). Like the program flow of a Minsky machine, a flowchart always starts at the top of a page and proceeds down. Its primary symbols are only four: the directed arrow showing program flow, the rectangle (SEQUENCE, GOTO), the diamond (IF-THEN-ELSE), and the dot (OR-tie). The Böhm–Jacopini canonical structures are made of these primitive shapes. Sub-structures can "nest" in rectangles, but only if a single exit occurs from the superstructure. The symbols, and their use to build the canonical structures are shown in the diagram.

Examples

Algorithm example

One of the simplest algorithms is to find the largest number in a list of numbers of random order. Finding the solution requires looking at every number in the list. From this follows a simple algorithm, which can be stated in a high-level description in English prose, as:

High-level description:

1. If there are no numbers in the set then there is no highest number.
2. Assume the first number in the set is the largest number in the set.
3. For each remaining number in the set: if this number is larger than the current largest number, consider this number to be the largest number in the set.
4. When there are no numbers left in the set to iterate over, consider the current largest number to be the largest number of the set.

(Quasi-)formal description: Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

```
Algorithm LargestNumber
Input: A list of numbers L.
Output: The largest number in the list L.
```

```
if L.size = 0 return null
largest ← L[0]
for each item in L, do
if item > largest, then
largest ← item
return largest
```

- "←" denotes assignment. For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

Euclid's algorithm

In mathematics, the **Euclidean algorithm**, or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC).[61] It is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

Euclid poses the problem thus: "Given two numbers not prime to one another, to find their greatest common measure". He defines "A number [to be] a multitude composed of units": a counting number, a positive integer not including zero. To "measure" is to place a shorter measuring length *s* successively (*q* times) along longer length *l* until the remaining portion *r* is less than the shorter length *s*.[62] In modern words, remainder $r = l - q \times s$, *q* being the quotient, or remainder *r* is the "modulus", the integer-fractional part left over after the division.[63]

For Euclid's method to succeed, the starting lengths must satisfy two requirements: (i) the lengths must not be zero, AND (ii) the subtraction must be "proper"; i.e., a test must guarantee that the smaller of the two numbers is subtracted from the larger (or the two can be equal so their subtraction yields zero).

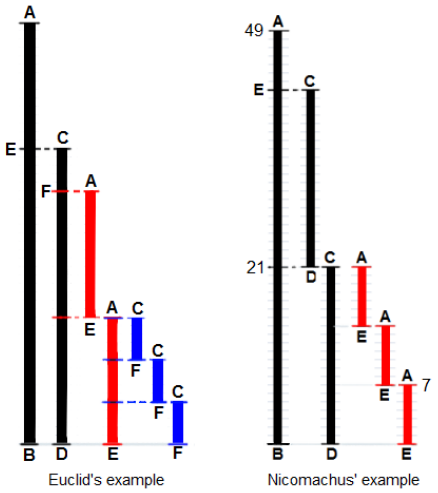
Euclid's original proof adds a third requirement: the two lengths must not be prime to one another. Euclid stipulated this so that he could construct a reductio ad absurdum proof that the two numbers' common measure is in fact the *greatest*.[64] While Nicomachus' algorithm is the same as Euclid's, when the numbers are prime to one another, it yields the number "1" for their common measure. So, to be precise, the following is really Nicomachus' algorithm.

Computer language for Euclid's algorithm

Only a few instruction *types* are required to execute Euclid's algorithm—some logical tests (conditional GOTO), unconditional GOTO, assignment (replacement), and subtraction.

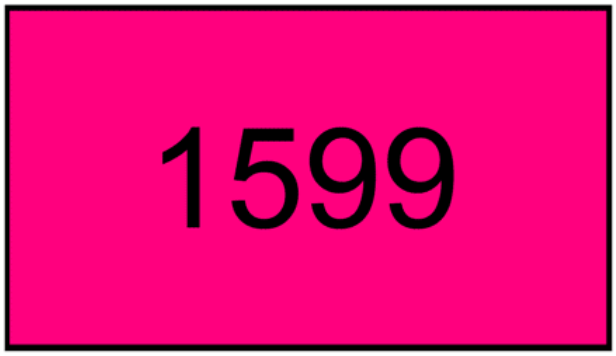
- A *location* is symbolized by upper case letter(s), e.g. S, A, etc.
- The varying quantity (number) in a location is written in lower case letter(s) and (usually) associated with the location's name. For example, location L at the start might contain the number *l* = 3009.

An



Euclid's example Nicomachus' example

The example-diagram of Euclid's algorithm from T.L. Heath (1908), with more detail added. Euclid does not go beyond a third measuring and gives no numerical examples. Nicomachus gives the example of 49 and 21: "I subtract the less from the greater; 28 is left; then again I subtract from this the same 21 (for this is possible); 7 is left; I subtract this from 21, 14 is left; from which I again subtract 7 (for this is possible); 7 is left, but 7 cannot be subtracted from 7." Heath comments that "The last phrase is curious, but the meaning of it is obvious enough, as also the meaning of the phrase about ending 'at one and the same number'."(Heath 1908:300).



A graphical expression of Euclid's algorithm to find the greatest common divisor for 1599 and 650.

```
1599 = 650×2 + 299
650 = 299×2 + 52
299 = 52×5 + 39
52 = 39×1 + 13
39 = 13×3 + 0
```

inelegant program for Euclid's algorithm

The following algorithm is framed as Knuth's four-step version of Euclid's and Nicomachus', but, rather than using division to find the remainder, it uses successive subtractions of the shorter length *s* from the remaining length *r* until *r* is less than *s*. The high-level description, shown in boldface, is adapted from Knuth 1973:2–4:

INPUT:

```
1 [Into two locations L and S put the numbers l and s that
represent the two lengths]:
INPUT L, S
2 [Initialize R: make the remaining length r equal to the
starting/initial/input length l]:
R ← L
```

E0: [Ensure *r* ≥ *s*.]

```
3 [Ensure the smaller of the two numbers is in S and the larger in R]:
IF R > S THEN
the contents of L is the larger number so skip over the exchange-steps 4, 5 and 6:
GOTO step 7
ELSE
swap the contents of R and S.
4 L ← R (this first step is redundant, but is useful for later discussion).
5 R ← S
6 S ← L
```

E1: [Find remainder]: Until the remaining length *r* in R is less than the shorter length *s* in S, repeatedly subtract the measuring number *s* in S from the remaining length *r* in R.

```
7 IF S > R THEN
done measuring so
GOTO 10
ELSE
```

```
measure again,
8 R ← R − S
9 [Remainder-loop]:
GOTO 7.
```

E2: [Is the remainder zero?]: EITHER (i) the last measure was exact, the remainder in R is zero, and the program can halt, OR (ii) the algorithm must continue: the last measure left a remainder in R less than measuring number in S.

```
10 IF R = 0 THEN
done so
GOTO step 15
ELSE
CONTINUE TO step 11,
```

E3: [Interchange *s* and *r*]: The nut of Euclid's algorithm. Use remainder *r* to measure what was previously smaller number *s*; L serves as a temporary location.

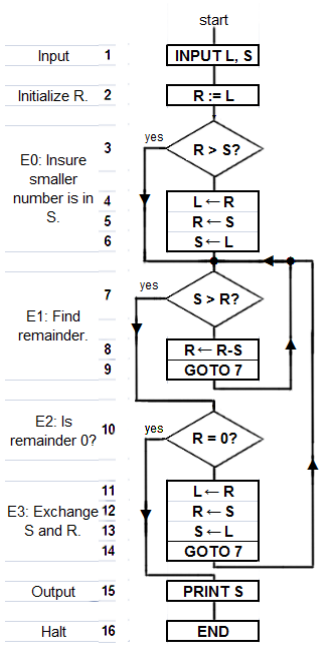
```
11 L ← R
12 R ← S
13 S ← L
14 [Repeat the measuring process]:
GOTO 7
```

OUTPUT:

```
15 [Done. S contains the greatest common divisor]:
PRINT S
```

DONE:

```
16 HALT, END, STOP.
```



"Inelegant"

"Inelegant" is a translation of Knuth's version of the algorithm with a subtraction-based remainder-loop replacing his use of division (or a "modulus" instruction). Derived from Knuth 1973:2–4. Depending on the two numbers "Inelegant" may compute the g.c.d. in fewer steps than "Elegant".

An elegant program for Euclid's algorithm

The following version of Euclid's algorithm requires only six core instructions to do what thirteen are required to do by "Inelegant"; worse, "Inelegant" requires more *types* of instructions. The flowchart of "Elegant" can be found at the top of this article. In the (unstructured) Basic language, the steps are numbered, and the instruction **LET** `[]` = `[]` is the assignment instruction symbolized by ←.

```
5 REM Euclid's algorithm for greatest common divisor
6 PRINT "Type two integers greater than 0"
10 INPUT A,B
20 IF B=0 THEN GOTO 80
30 IF A > B THEN GOTO 60
40 LET B=B-A
50 GOTO 20
60 LET A=A-B
70 GOTO 20
80 PRINT A
90 END
```

How "Elegant" works: In place of an outer "Euclid loop", "Elegant" shifts back and forth between two "co-loops", an A > B loop that computes A ← A − B, and a B ≤ A loop that computes B ← B − A. This works because, when at last the minuend M is less than or equal to the subtrahend S (Difference = Minuend − Subtrahend), the minuend can become *s* (the new measuring length) and the subtrahend can become the new *r* (the length to be measured); in other words the "sense" of the subtraction reverses.

The following version can be used with programming languages from the C-family:

```
// Euclid's algorithm for greatest common divisor
int euclidAlgorithm (int A, int B){
     A=abs(A);
     B=abs(B);
     while (B!=0){
          while (A>B) A=A-B;
          B=B-A;
     }
     return A;
}
```

Testing the Euclid algorithms

Does an algorithm do what its author wants it to do? A few test cases usually give some confidence in the core functionality. But tests are not enough. For test cases, one source[65] uses 3009 and 884. Knuth suggested 40902, 24140. Another interesting case is the two relatively prime numbers 14157 and 5950.

But "exceptional cases"[66] must be identified and tested. Will "Inelegant" perform properly when R > S, S > R, R = S? Ditto for "Elegant": B > A, A > B, A = B? (Yes to all). What happens when one number is zero, both numbers are zero? ("Inelegant" computes forever in all cases; "Elegant" computes forever when A = 0.) What happens if *negative* numbers are entered? Fractional numbers? If the input numbers, i.e. the domain of the function computed by the algorithm/program, is to include only positive integers including zero, then the failures at zero indicate that the algorithm (and the program that instantiates it) is a partial function rather than a total function. A notable failure due to exceptions is the Ariane 5 Flight 501 rocket failure (June 4, 1996).

Proof of program correctness by use of mathematical induction: Knuth demonstrates the application of mathematical induction to an "extended" version of Euclid's algorithm, and he proposes "a general method applicable to proving the validity of any algorithm".[67] Tausworthe proposes that a measure of the complexity of a program be the length of its correctness proof.[68]

Measuring and improving the Euclid algorithms

Elegance (compactness) versus goodness (speed): With only six core instructions, "Elegant" is the clear winner, compared to "Inelegant" at thirteen instructions. However, "Inelegant" is *faster* (it arrives at HALT in fewer steps). Algorithm analysis[69] indicates why this is the case: "Elegant" does *two* conditional tests in every subtraction loop, whereas "Inelegant" only does one. As the algorithm (usually) requires many loop-throughs, *on average* much time is wasted doing a "B = 0?" test that is needed only after the remainder is computed.

Can the algorithms be improved?: Once the programmer judges a program "fit" and "effective"—that is, it computes the function intended by its author—then the question becomes, can it be improved?

The compactness of "Inelegant" can be improved by the elimination of five steps. But Chaitin proved that compacting an algorithm cannot be automated by a generalized algorithm;[70] rather, it can only be done heuristically; i.e., by exhaustive search (examples to be found at Busy beaver), trial and error, cleverness, insight, application of inductive reasoning, etc. Observe that steps 4, 5 and 6 are repeated in steps 11, 12 and 13. Comparison with "Elegant" provides a hint that these steps, together with steps 2 and 3, can be eliminated. This reduces the number of core instructions from thirteen to eight, which makes it "more elegant" than "Elegant", at nine steps.

The speed of "Elegant" can be improved by moving the "B=0?" test outside of the two subtraction loops. This change calls for the addition of three instructions (B = 0?, A = 0?, GOTO). Now "Elegant" computes the example-numbers faster; whether this is always the case for any given A, B, and R, S would require a detailed analysis.

Algorithmic analysis

It is frequently important to know how much of a particular resource (such as time or storage) is theoretically required for a given algorithm. Methods have been developed for the analysis of algorithms to obtain such quantitative answers (estimates); for example, an algorithm which adds up the elements of a list of n numbers would have a time requirement of $O(n)$, using big O notation. At all times the algorithm only needs to remember two values: the sum of all the elements so far, and its current position in the input list. Therefore, it is said to have a space requirement of $O(1)$, if the space required to store the input numbers is not counted, or $O(n)$ if it is counted.

Different algorithms may complete the same task with a different set of instructions in less or more time, space, or 'effort' than others. For example, a binary search algorithm (with cost $O(\log n)$) outperforms a sequential search (cost $O(n)$) when used for table lookups on sorted lists or arrays.

Formal versus empirical

The analysis, and study of algorithms is a discipline of computer science, and is often practiced abstractly without the use of a specific programming language or implementation. In this sense, algorithm analysis resembles other mathematical disciplines in that it focuses on the underlying properties of the algorithm and not on the specifics of any particular implementation. Usually pseudocode is used for analysis as it is the simplest and most general representation. However, ultimately, most algorithms are usually implemented on particular hardware/software platforms and their algorithmic efficiency is eventually put to the test using real code. For the solution of a "one off" problem, the efficiency of a particular algorithm may not have significant consequences (unless n is extremely large) but for algorithms designed for fast interactive, commercial or long life scientific usage it may be critical. Scaling from small n to large n frequently exposes inefficient algorithms that are otherwise benign.

Empirical testing is useful because it may uncover unexpected interactions that affect performance. Benchmarks may be used to compare before/after potential improvements to an algorithm after program optimization. Empirical tests cannot replace formal analysis, though, and are not trivial to perform in a fair manner.[71]

Execution efficiency

To illustrate the potential improvements possible even in well-established algorithms, a recent significant innovation, relating to FFT algorithms (used heavily in the field of image processing), can decrease processing time up to 1,000 times for applications like medical imaging.[72] In general, speed improvements depend on special properties of the problem, which are very common

in practical applications.[73] Speedups of this magnitude enable computing devices that make extensive use of image processing (like digital cameras and medical equipment) to consume less power.

Classification

There are various ways to classify algorithms, each with its own merits.

By implementation

One way to classify algorithms is by implementation means.

Recursion

A recursive algorithm is one that invokes (makes reference to) itself repeatedly until a certain condition (also known as termination condition) matches, which is a method common to functional programming. Iterative algorithms use repetitive constructs like loops and sometimes additional data structures like stacks to solve the given problems. Some problems are naturally suited for one implementation or the other. For example, towers of Hanoi is well understood using recursive implementation. Every recursive version has an equivalent (but possibly more or less complex) iterative version, and vice versa.

```
int gcd(int A, int B) {
    if (B == 0)
        return A;
    else if (A > B)
        return gcd(A-B,B);
    else
        return gcd(A,B-A);
}
```

Recursive C implementation of Euclid's algorithm from the above flowchart

Logical

An algorithm may be viewed as controlled logical deduction. This notion may be expressed as: *Algorithm = logic + control*.[74] The logic component expresses the axioms that may be used in the computation and the control component determines the way in which deduction is applied to the axioms. This is the basis for the logic programming paradigm. In pure logic programming languages, the control component is fixed and algorithms are specified by supplying only the logic component. The appeal of this approach is the elegant semantics: a change in the axioms produces a well-defined change in the algorithm.

Serial, parallel or distributed

Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time. Those computers are sometimes called serial computers. An algorithm designed for such an environment is called a serial algorithm, as opposed to parallel algorithms or distributed algorithms. Parallel algorithms take advantage of computer architectures where several processors can work on a problem at the same time, whereas distributed algorithms utilize multiple machines connected with a computer network. Parallel or distributed algorithms divide the problem into more symmetrical or asymmetrical subproblems and collect the results back together. The resource consumption in such algorithms is not only processor cycles on each processor but also the communication overhead between the processors. Some sorting algorithms can be parallelized efficiently, but their communication overhead is expensive. Iterative algorithms are generally parallelizable. Some problems have no parallel algorithms and are called inherently serial problems.

Deterministic or non-deterministic

Deterministic algorithms solve the problem with exact decision at every step of the algorithm whereas non-deterministic algorithms solve problems via guessing although typical guesses are made more accurate through the use of heuristics.

Exact or approximate

While many algorithms reach an exact solution, approximation algorithms seek an approximation that is closer to the true solution. The approximation can be reached by either using a deterministic or a random strategy. Such algorithms have practical value for many

hard problems. One of the examples of an approximate algorithm is the Knapsack problem, where there is a set of given items. Its goal is to pack the knapsack to get the maximum total value. Each item has some weight and some value. Total weight that can be carried is no more than some fixed number X. So, the solution must consider weights of items as well as their value.[75]

Quantum algorithm

They run on a realistic model of quantum computation. The term is usually used for those algorithms which seem inherently quantum, or use some essential feature of Quantum computing such as quantum superposition or quantum entanglement.

By design paradigm

Another way of classifying algorithms is by their design methodology or paradigm. There is a certain number of paradigms, each different from the other. Furthermore, each of these categories includes many different types of algorithms. Some common paradigms are:

Brute-force or exhaustive search

This is the naive method of trying every possible solution to see which is best.[76]

Divide and conquer

A divide-and-conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments. A simpler variant of divide and conquer is called a *decrease-and-conquer algorithm*, which solves an identical subproblem and uses the solution of this subproblem to solve the bigger problem. Divide and conquer divides the problem into multiple subproblems and so the conquer stage is more complex than decrease and conquer algorithms. An example of a decrease and conquer algorithm is the binary search algorithm.

Search and enumeration

Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.

Randomized algorithm

Such algorithms make some choices randomly (or pseudo-randomly). They can be very useful in finding approximate solutions for problems where finding exact solutions can be impractical (see heuristic method below). For some of these problems, it is known that the fastest approximations must involve some randomness.[77] Whether randomized algorithms with polynomial time complexity can be the fastest algorithms for some problems is an open question known as the P versus NP problem. There are two large classes of such algorithms:

1. Monte Carlo algorithms return a correct answer with high-probability. E.g. RP is the subclass of these that run in polynomial time.
2. Las Vegas algorithms always return the correct answer, but their running time is only probabilistically bound, e.g. ZPP.

Reduction of complexity

This technique involves solving a difficult problem by transforming it into a better-known problem for which we have (hopefully) asymptotically optimal algorithms. The goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithm's. For example, one selection algorithm for finding the median in an unsorted list involves first sorting the list (the expensive portion) and then pulling out the middle element in the sorted list (the cheap portion). This technique is also known as *transform and conquer*.

Back tracking

In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution.

Optimization problems

For optimization problems there is a more specific classification of algorithms; an algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

Linear programming

When searching for optimal solutions to a linear function bound to linear equality and inequality constraints, the constraints of the problem can be used directly in producing the optimal solutions. There are algorithms that can solve any problem in this category, such as the popular simplex algorithm.[78] Problems that can be solved with linear programming include the maximum flow problem for directed graphs. If a problem additionally requires that one or more of the unknowns must be an integer then it is classified in integer programming. A linear programming algorithm can solve such a problem if it can be proved that all restrictions for integer values are superficial, i.e., the solutions satisfy these restrictions anyway. In the general case, a specialized algorithm or an algorithm that finds approximate solutions is used, depending on the difficulty of the problem.

Dynamic programming

When a problem shows optimal substructures—meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems—and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called *dynamic programming* avoids recomputing solutions that have already been computed. For example, Floyd–Warshall algorithm, the shortest path to a goal from a vertex in a weighted graph can be found by using the shortest path to the goal from all adjacent vertices. Dynamic programming and memoization go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming. The difference between dynamic programming and straightforward recursion is in caching or memoization of recursive calls. When subproblems are independent and there is no repetition, memoization does not help; hence dynamic programming is not a solution for all complex problems. By using memoization or maintaining a table of subproblems already solved, dynamic programming reduces the exponential nature of many problems to polynomial complexity.

The greedy method

A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution, which may be given or have been constructed in some way, and improve it by making small modifications. For some problems they can find the optimal solution while for others they stop at local optima, that is, at solutions that cannot be improved by the algorithm but are not optimum. The most popular use of greedy algorithms is for finding the minimal spanning tree where finding the optimal solution is possible with this method. Huffman Tree, Kruskal, Prim, Sollin are greedy algorithms that can solve this optimization problem.

The heuristic method

In optimization problems, heuristic algorithms can be used to find a solution close to the optimal solution in cases where finding the optimal solution is impractical. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Their merit is that they can find a solution very close to the optimal solution in a relatively short time. Such algorithms include local search, tabu search, simulated annealing, and genetic algorithms. Some of them, like simulated annealing, are non-deterministic algorithms while others, like tabu search, are deterministic. When a bound on the error of the non-optimal solution is known, the algorithm is further categorized as an approximation algorithm.

By field of study

Every field of science has its own problems and needs efficient algorithms. Related problems in one field are often studied together. Some example classes are search algorithms, sorting algorithms, merge algorithms, numerical algorithms, graph algorithms, string algorithms, computational geometric algorithms, combinatorial algorithms, medical algorithms, machine learning, cryptography, data compression algorithms and parsing techniques.

Fields tend to overlap with each other, and algorithm advances in one field may improve those of other, sometimes completely unrelated, fields. For example, dynamic programming was invented for optimization of resource consumption in industry but is now used in solving a broad range of problems in many fields.

By complexity

Algorithms can be classified by the amount of time they need to complete compared to their input size:

- Constant time: if the time needed by the algorithm is the same, regardless of the input size. E.g. an access to an array element.
- Logarithmic time: if the time is a logarithmic function of the input size. E.g. binary search algorithm.
- Linear time: if the time is proportional to the input size. E.g. the traverse of a list.
- Polynomial time: if the time is a power of the input size. E.g. the bubble sort algorithm has quadratic time complexity.
- Exponential time: if the time is an exponential function of the input size. E.g. Brute-force search.

Some problems may have multiple algorithms of differing complexity, while other problems might have no algorithms or no known efficient algorithms. There are also mappings from some problems to other problems. Owing to this, it was found to be more suitable to classify the problems themselves instead of the algorithms into equivalence classes based on the complexity of the best possible algorithms for them.

Continuous algorithms

The adjective "continuous" when applied to the word "algorithm" can mean:

- An algorithm operating on data that represents continuous quantities, even though this data is represented by discrete approximations—such algorithms are studied in numerical analysis; or
- An algorithm in the form of a differential equation that operates continuously on the data, running on an analog computer.[79]

Legal issues

Algorithms, by themselves, are not usually patentable. In the United States, a claim consisting solely of simple manipulations of abstract concepts, numbers, or signals does not constitute "processes" (USPTO 2006), and hence algorithms are not patentable (as in Gottschalk v. Benson). However practical applications of algorithms are sometimes patentable. For example, in Diamond v. Diehr, the application of a simple feedback algorithm to aid in the curing of synthetic rubber was

deemed patentable. The patenting of software is highly controversial, and there are highly criticized patents involving algorithms, especially data compression algorithms, such as Unisys' LZW patent.

Additionally, some cryptographic algorithms have export restrictions (see export of cryptography).

History: Development of the notion of "algorithm"

Ancient Near East

The earliest evidence of algorithms is found in the Babylonian mathematics of ancient Mesopotamia (modern Iraq). A Sumerian clay tablet found in Shuruppak near Baghdad and dated to circa 2500 BC described the earliest division algorithm.[11] During the Hammurabi dynasty circa 1800-1600 BC, Babylonian clay tablets described algorithms for computing formulas.[80] Algorithms were also used in Babylonian astronomy. Babylonian clay tablets describe and employ algorithmic procedures to compute the time and place of significant astronomical events.[81]

Algorithms for arithmetic are also found in ancient Egyptian mathematics, dating back to the Rhind Mathematical Papyrus circa 1550 BC.[11] Algorithms were later used in ancient Hellenistic mathematics. Two examples are the Sieve of Eratosthenes, which was described in the *Introduction to Arithmetic* by Nicomachus,[82][12]:Ch 9.2 and the Euclidean algorithm, which was first described in *Euclid's Elements* (*c.* 300 BC).[12]:Ch 9.1

Discrete and distinguishable symbols

Tally-marks: To keep track of their flocks, their sacks of grain and their money the ancients used tallying: accumulating stones or marks scratched on sticks or making discrete symbols in clay. Through the Babylonian and Egyptian use of marks and symbols, eventually Roman numerals and the abacus evolved (Dilson, p. 16–41). Tally marks appear prominently in unary numeral system arithmetic used in Turing machine and Post–Turing machine computations.

Manipulation of symbols as "place holders" for numbers: algebra

Muhammad ibn Mūsā al-Khwārizmī, a Persian mathematician, wrote the *Al-jabr* in the 9th century. The terms "algorism" and "algorithm" are derived from the name al-Khwārizmī, while the term "algebra" is derived from the book *Al-jabr*. In Europe, the word "algorithm" was originally used to refer to the sets of rules and techniques used by Al-Khwarizmi to solve algebraic equations, before later being generalized to refer to any set of rules or techniques.[83] This eventually culminated in Leibniz's notion of the calculus ratiocinator (*c.* 1680):

> A good century and a half ahead of his time, Leibniz proposed an algebra of logic, an algebra that would specify the rules for manipulating logical concepts in the manner that ordinary algebra specifies the rules for manipulating numbers.[84]

Cryptographic algorithms

The first cryptographic algorithm for deciphering encrypted code was developed by Al-Kindi, a 9th-century Arab mathematician, in *A Manuscript On Deciphering Cryptographic Messages*. He gave the first description of cryptanalysis by frequency analysis, the earliest codebreaking algorithm.[13]

Mechanical contrivances with discrete states

The clock: Bolter credits the invention of the weight-driven clock as "The key invention [of Europe in the Middle Ages]", in particular, the verge escapement[85] that provides us with the tick and tock of a mechanical clock. "The accurate automatic machine"[86] led immediately to "mechanical automata" beginning in the 13th century and finally to "computational machines"—the difference engine and analytical engines of Charles Babbage and Countess Ada Lovelace, mid-19th century.[87] Lovelace is credited with the first creation of an algorithm intended for processing on a computer—Babbage's analytical engine, the first device considered a real Turing-complete computer instead of just a calculator—and is sometimes called "history's first programmer" as a result, though a full implementation of Babbage's second device would not be realized until decades after her lifetime.

Logical machines 1870 – Stanley Jevons' "logical abacus" and "logical machine": The technical problem was to reduce Boolean equations when presented in a form similar to what is now known as Karnaugh maps. Jevons (1880) describes first a simple "abacus" of "slips of wood furnished with pins, contrived so that any part or class of the [logical] combinations can be picked out mechanically ... More recently, however, I have reduced the system to a completely mechanical form, and have thus embodied the whole of the indirect process of inference in what may be called a *Logical Machine*" His machine came equipped with "certain moveable wooden rods" and "at the foot are 21 keys like those of a piano [etc.] ...". With this machine he could analyze a "syllogism or any other simple logical argument".[88]

This machine he displayed in 1870 before the Fellows of the Royal Society.[89] Another logician John Venn, however, in his 1881 *Symbolic Logic*, turned a jaundiced eye to this effort: "I have no high estimate myself of the interest or importance of what are sometimes called logical machines ... it does not seem to me that any contrivances at present known or likely to be discovered really deserve the name of logical machines"; see more at Algorithm characterizations. But not to be outdone he too presented "a plan somewhat analogous, I apprehend, to Prof. Jevon's *abacus* ... [And] [a]gain, corresponding to Prof. Jevons's logical machine, the following contrivance may be described. I prefer to call it merely a logical-diagram machine ... but I suppose that it could do very completely all that can be rationally expected of any logical machine".[90]

Jacquard loom, Hollerith punch cards, telegraphy and telephony – the electromechanical relay: Bell and Newell (1971) indicate that the Jacquard loom (1801), precursor to Hollerith cards (punch cards, 1887), and "telephone switching technologies" were the roots of a tree leading to the development of the first computers.[91] By the mid-19th century the telegraph, the precursor of the telephone, was in use throughout the world, its discrete and distinguishable encoding of letters as "dots and dashes" a common sound. By the late 19th century the ticker tape (*c.* 1870s) was in use, as was the use of Hollerith cards in the 1890 U.S. census. Then came the teleprinter (*c.* 1910) with its punched-paper use of Baudot code on tape.

Telephone-switching networks of electromechanical relays (invented 1835) was behind the work of George Stibitz (1937), the inventor of the digital adding device. As he worked in Bell Laboratories, he observed the "burdensome' use of mechanical calculators with gears. "He went home one evening in 1937 intending to test his idea... When the tinkering was over, Stibitz had constructed a binary adding device".[92]

The mathematician Martin Davis observes the particular importance of the electromechanical relay (with its two "binary states" *open* and *closed*):

> It was only with the development, beginning in the 1930s, of electromechanical calculators using electrical relays, that machines were built having the scope Babbage had envisioned."[93]

Mathematics during the 19th century up to the mid-20th century

Symbols and rules: In rapid succession, the mathematics of George Boole (1847, 1854), Gottlob Frege (1879), and Giuseppe Peano (1888–1889) reduced arithmetic to a sequence of symbols manipulated by rules. Peano's *The principles of arithmetic, presented by a new method* (1888) was "the first attempt at an axiomatization of mathematics in a symbolic language".[94]

But Heijenoort gives Frege (1879) this kudos: Frege's is "perhaps the most important single work ever written in logic. ... in which we see a " 'formula language', that is a *lingua characterica*, a language written with special symbols, "for pure thought", that is, free from rhetorical embellishments ... constructed from specific symbols that are manipulated according to definite rules".[95] The work of Frege was further simplified and amplified by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (1910–1913).

The paradoxes: At the same time a number of disturbing paradoxes appeared in the literature, in particular, the Burali-Forti paradox (1897), the Russell paradox (1902–03), and the Richard Paradox.[96] The resultant considerations led to Kurt Gödel's paper (1931)—he specifically cites the paradox of the liar—that completely reduces rules of recursion to numbers.

Effective calculability: In an effort to solve the Entscheidungsproblem defined precisely by Hilbert in 1928, mathematicians first set about to define what was meant by an "effective method" or "effective calculation" or "effective calculability" (i.e., a calculation that would succeed). In rapid succession the following appeared: Alonzo Church, Stephen Kleene and J.B. Rosser's λ-calculus[97] a finely honed definition of "general recursion" from the work of Gödel acting on suggestions of Jacques Herbrand (cf. Gödel's Princeton lectures of 1934) and subsequent simplifications by Kleene.[98] Church's proof[99] that the Entscheidungsproblem was unsolvable, Emil Post's definition of effective calculability as a worker mindlessly following a list of instructions to move left or right through a sequence of rooms and while there either mark or erase a paper or observe the paper and make a yes-no decision about the next instruction.[100] Alan Turing's proof of that the Entscheidungsproblem was unsolvable by use of his "a- [automatic-] machine"[101]—in effect almost identical to Post's "formulation", J. Barkley Rosser's definition of "effective method" in terms of "a machine".[102] Kleene's proposal of a precursor to "Church thesis" that he called "Thesis I",[103] and a few years later Kleene's renaming his Thesis "Church's Thesis"[104] and proposing "Turing's Thesis".[105]

Emil Post (1936) and Alan Turing (1936–37, 1939)

Emil Post (1936) described the actions of a "computer" (human being) as follows:

> "...two concepts are involved: that of a *symbol space* in which the work leading from problem to answer is to be carried out, and a fixed unalterable *set of directions*.

His symbol space would be

> "a two-way infinite sequence of spaces or boxes... The problem solver or worker is to move and work in this symbol space, being capable of being in, and operating in but one box at a time.... a box is to admit of but two possible conditions, i.e., being empty or unmarked, and having a single mark in it, say a vertical stroke.
>
> "One box is to be singled out and called the starting point. ...a specific problem is to be given in symbolic form by a finite number of boxes [i.e., INPUT] being marked with a stroke. Likewise, the answer [i.e., OUTPUT] is to be given in symbolic form by such a configuration of marked boxes...

"A set of directions applicable to a general problem sets up a deterministic process when applied to each specific problem. This process terminates only when it comes to the direction of type (C) [i.e., STOP]".[106] See more at Post–Turing machine

Alan Turing's work[107] preceded that of Stibitz (1937); it is unknown whether Stibitz knew of the work of Turing. Turing's biographer believed that Turing's use of a typewriter-like model derived from a youthful interest: "Alan had dreamt of inventing typewriters as a boy; Mrs. Turing had a typewriter, and he could well have begun by asking himself what was meant by calling a typewriter 'mechanical'".[108] Given the prevalence at the time of Morse code, telegraphy, ticker tape machines, and teletypewriters, it is quite possible that all were influences on Turing during his youth.

Alan Turing's statue at Bletchley Park

Turing—his model of computation is now called a Turing machine—begins, as did Post, with an analysis of a human computer that he whittles down to a simple set of basic motions and "states of mind". But he continues a step further and creates a machine as a model of computation of numbers.[109]

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book...I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite...

"The behavior of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares that the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite...

"Let us imagine that the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided."[110]

Turing's reduction yields the following:

"The simple operations must therefore include:

"(a) Changes of the symbol on one of the observed squares
"(b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

"It may be that some of these change necessarily invoke a change of state of mind. The most general single operation must, therefore, be taken to be one of the following:

"(A) A possible change (a) of symbol together with a possible change of state of mind.
"(B) A possible change (b) of observed squares, together with a possible change of state of mind"

"We may now construct a machine to do the work of this computer."[110]

A few years later, Turing expanded his analysis (thesis, definition) with this forceful expression of it:

"A function is said to be "effectively calculable" if its values can be found by some purely mechanical process. Though it is fairly easy to get an intuitive grasp of this idea, it is nevertheless desirable to have some more definite, mathematical expressible definition ... [he

discusses the history of the definition pretty much as presented above with respect to Gödel, Herbrand, Kleene, Church, Turing, and Post] ... We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and to an identification of computability † with effective calculability...

> "† We shall use the expression "computable function" to mean a function calculable by a machine, and we let "effectively calculable" refer to the intuitive idea without particular identification with any one of these definitions".[111]

J.B. Rosser (1939) and S.C. Kleene (1943)

J. Barkley Rosser defined an 'effective [mathematical] method' in the following manner (italicization added):

> "'Effective method' is used here in the rather special sense of a method each step of which is precisely determined and which is certain to produce the answer in a finite number of steps. With this special meaning, three different precise definitions have been given to date. [his footnote #5; see discussion immediately below]. The simplest of these to state (due to Post and Turing) says essentially that *an effective method of solving certain sets of problems exists if one can build a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer*. All three definitions are equivalent, so it doesn't matter which one is used. Moreover, the fact that all three are equivalent is a very strong argument for the correctness of any one." (Rosser 1939:225–226)

Rosser's footnote No. 5 references the work of (1) Church and Kleene and their definition of λ-definability, in particular, Church's use of it in his *An Unsolvable Problem of Elementary Number Theory* (1936); (2) Herbrand and Gödel and their use of recursion, in particular, Gödel's use in his famous paper *On Formally Undecidable Propositions of Principia Mathematica and Related Systems I* (1931); and (3) Post (1936) and Turing (1936–37) in their mechanism-models of computation.

Stephen C. Kleene defined as his now-famous "Thesis I" known as the Church–Turing thesis. But he did this in the following context (boldface in original):

> "12. *Algorithmic theories*... In setting up a complete algorithmic theory, what we do is to describe a procedure, performable for each set of values of the independent variables, which procedure necessarily terminates and in such manner that from the outcome we can read a definite answer, "yes" or "no," to the question, "is the predicate value true?"" (Kleene 1943:273)

History after 1950

A number of efforts have been directed toward further refinement of the definition of "algorithm", and activity is on-going because of issues surrounding, in particular, foundations of mathematics (especially the Church–Turing thesis) and philosophy of mind (especially arguments about artificial intelligence). For more, see Algorithm characterizations.

See also

- Abstract machine
- Algorithm engineering
- Algorithm characterizations

- Algorithmic bias
- Algorithmic composition
- Algorithmic entities
- Algorithmic synthesis
- Algorithmic technique
- Algorithmic topology
- Garbage in, garbage out
- *Introduction to Algorithms* (textbook)
- Government by algorithm
- List of algorithms
- List of algorithm general topics
- List of important publications in theoretical computer science – Algorithms
- Regulation of algorithms
- Theory of computation
  - Computability theory
  - Computational complexity theory
- Computational mathematics

Notes

1. "Definition of ALGORITHM" (https://www.merriam-webster.com/dictionary/algorithm). *Merriam-Webster Online Dictionary*. Archived (https://web.archive.org/web/20200214074446/https://www.merriam-webster.com/dictionary/algorithm) from the original on February 14, 2020. Retrieved November 14, 2019.
2. Blair, Ann, Duguid, Paul, Goeing, Anja-Silvia and Grafton, Anthony. Information: A Historical Companion, Princeton: Princeton University Press, 2021. p. 247
3. David A. Grossman, Ophir Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd edition, 2004, ISBN 1402030045
4. "Any classical mathematical algorithm, for example, can be described in a finite number of English words" (Rogers 1987:2).
5. Well defined with respect to the agent that executes the algorithm: "There is a computing agent, usually human, which can react to the instructions and carry out the computations" (Rogers 1987:2).
6. "an algorithm is a procedure for computing a *function* (with respect to some chosen notation for integers) ... this limitation (to numerical functions) results in no loss of generality", (Rogers 1987:1).
7. "An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins" (Knuth 1973:5).
8. "A procedure which has all the characteristics of an algorithm except that it possibly lacks finiteness may be called a 'computational method'" (Knuth 1973:5).
9. "An algorithm has one or more outputs, i.e. quantities which have a specified relation to the inputs" (Knuth 1973:5).
10. Whether or not a process with random interior processes (not including the input) is an algorithm is debatable. Rogers opines that: "a computation is carried out in a discrete stepwise fashion, without the use of continuous methods or analogue devices ... carried forward deterministically, without resort to random methods or devices, e.g., dice" (Rogers 1987:2).
11. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*. Springer Science & Business Media. pp. 7–8. ISBN 9783642181924.
12. Cooke, Roger L. (2005). *The History of Mathematics: A Brief Course*. John Wiley & Sons. ISBN 978-1-118-46029-0.

13. Dooley, John F. (2013). *A Brief History of Cryptology and Cryptographic Algorithms*. Springer Science & Business Media. pp. 12–3. ISBN 9783319016283.
14. "Al-Khwarizmi biography" (http://www-history.mcs.st-andrews.ac.uk/Biographies/Al-Khwarizmi.html). *www-history.mcs.st-andrews.ac.uk*. Archived (https://web.archive.org/web/20190802091553/http://www-history.mcs.st-andrews.ac.uk/Biographies/Al-Khwarizmi.html) from the original on August 2, 2019. Retrieved May 3, 2017.
15. "Etymology of algorithm" (http://chambers.co.uk/search/?query=algorithm&title=21st). *Chambers Dictionary*. Archived (https://web.archive.org/web/20190331204600/https://chambers.co.uk/search/?query=algorithm&title=21st) from the original on March 31, 2019. Retrieved December 13, 2016.
16. Hogendijk, Jan P. (1998). "al-Khwarzimi" (https://web.archive.org/web/20090412193516/http://www.kennislink.nl/web/show?id=116543). *Pythagoras*. **38** (2): 4–5. Archived from the original (http://www.kennislink.nl/web/show?id=116543) on April 12, 2009.
17. Oaks, Jeffrey A. "Was al-Khwarizmi an applied algebraist?" (https://web.archive.org/web/20110718094835/http://facstaff.uindy.edu/~oaks/MHMC.htm). University of Indianapolis. Archived from the original (http://facstaff.uindy.edu/~oaks/MHMC.htm) on July 18, 2011. Retrieved May 30, 2008.
18. Brezina, Corona (2006). *Al-Khwarizmi: The Inventor Of Algebra* (https://books.google.com/books?id=955jPgAACAAJ). The Rosen Publishing Group. ISBN 978-1-4042-0513-0.
19. Foremost mathematical texts in history (http://www-history.mcs.st-and.ac.uk/Extras/Boyer_Foremost_Text.html) Archived (https://web.archive.org/web/20110609224820/http://www-history.mcs.st-and.ac.uk/Extras/Boyer_Foremost_Text.html) June 9, 2011, at the Wayback Machine, according to Carl B. Boyer.
20. "algorismic" (https://www.thefreedictionary.com/algorismic), *The Free Dictionary*, archived (https://web.archive.org/web/20191221200124/https://www.thefreedictionary.com/algorismic) from the original on December 21, 2019, retrieved November 14, 2019
21. *Oxford English Dictionary*, Third Edition, 2012 *s.v.* (http://www.oed.com/view/Entry/4959)
22. Sriram, M. S. (2005). "Algorithms in Indian Mathematics" (https://books.google.com/books?id=qfJdDwAAQBAJ&pg=PA153). In Emch, Gerard G.; Sridharan, R.; Srinivas, M. D. (eds.). *Contributions to the History of Indian Mathematics*. Springer. p. 153. ISBN 978-93-86279-25-5.
23. Mehri, Bahman (2017). "From Al-Khwarizmi to Algorithm". *Olympiads in Informatics*. **11** (2): 71–74. doi:10.15388/ioi.2017.special.11 (https://doi.org/10.15388%2Fioi.2017.special.11).
24. "Abu Jafar Muhammad ibn Musa al-Khwarizmi" (http://members.peak.org/~jeremy/calculators/alKwarizmi.html). *members.peak.org*. Archived (https://web.archive.org/web/20190821232118/http://members.peak.org/~jeremy/calculators/alKwarizmi.html) from the original on August 21, 2019. Retrieved November 14, 2019.
25. Kleene 1943 in Davis 1965:274
26. Rosser 1939 in Davis 1965:225
27. Stone 1973:4
28. Simanowski, Roberto (2018). *The Death Algorithm and Other Digital Dilemmas* (https://books.google.com/books?id=RJV5DwAAQBAJ). Untimely Meditations. Vol. 14. Translated by Chase, Jefferson. Cambridge, Massachusetts: MIT Press. p. 147. ISBN 9780262536370. Archived (https://web.archive.org/web/20191222120705/https://books.google.com/books?id=RJV5DwAAQBAJ) from the original on December 22, 2019. Retrieved May 27, 2019. "[...] the next level of abstraction of central bureaucracy: globally operating algorithms."
29. Dietrich, Eric (1999). "Algorithm". In Wilson, Robert Andrew; Keil, Frank C. (eds.). *The MIT Encyclopedia of the Cognitive Sciences* (https://books.google.com/books?id=-wt1aZrGXLYC). MIT Cognet library. Cambridge, Massachusetts: MIT Press (published 2001). p. 11. ISBN 9780262731447. Retrieved July 22, 2020. "An algorithm is a recipe, method, or technique for doing something."
30. Stone simply requires that "it must terminate in a finite number of steps" (Stone 1973:7–8).
31. Boolos and Jeffrey 1974,1999:19

32. cf Stone 1972:5
33. Knuth 1973:7 states: "In practice we not only want algorithms, we want *good* algorithms ... one criterion of goodness is the length of time taken to perform the algorithm ... other criteria are the adaptability of the algorithm to computers, its simplicity, and elegance, etc."
34. cf Stone 1973:6
35. Stone 1973:7–8 states that there must be, "...a procedure that a robot [i.e., computer] can follow in order to determine precisely how to obey the instruction". Stone adds finiteness of the process, and definiteness (having no ambiguity in the instructions) to this definition.
36. Knuth, loc. cit
37. Minsky 1967, p. 105
38. Gurevich 2000:1, 3
39. Sipser 2006:157
40. Goodrich, Michael T.; Tamassia, Roberto (2002), *Algorithm Design: Foundations, Analysis, and Internet Examples* (http://ww3.algorithmdesign.net/ch00-front.html), John Wiley & Sons, Inc., ISBN 978-0-471-38365-9, archived (https://web.archive.org/web/20150428201622/http://ww3.algorithmdesign.net/ch00-front.html) from the original on April 28, 2015, retrieved June 14, 2018
41. Knuth 1973:7
42. Chaitin 2005:32
43. Rogers 1987:1–2
44. In his essay "Calculations by Man and Machine: Conceptual Analysis" Seig 2002:390 credits this distinction to Robin Gandy, cf Wilfred Seig, et al., 2002 *Reflections on the foundations of mathematics: Essays in honor of Solomon Feferman*, Association for Symbolic Logic, A.K. Peters Ltd, Natick, MA.
45. cf Gandy 1980:126, Robin Gandy *Church's Thesis and Principles for Mechanisms* appearing on pp. 123–148 in J. Barwise et al. 1980 *The Kleene Symposium*, North-Holland Publishing Company.
46. A "robot": "A computer is a robot that performs any task that can be described as a sequence of instructions." cf Stone 1972:3
47. Lambek's "abacus" is a "countably infinite number of locations (holes, wires etc.) together with an unlimited supply of counters (pebbles, beads, etc.). The locations are distinguishable, the counters are not". The holes have unlimited capacity, and standing by is an agent who understands and is able to carry out the list of instructions" (Lambek 1961:295). Lambek references Melzak who defines his Q-machine as "an indefinitely large number of locations ... an indefinitely large supply of counters distributed among these locations, a program, and an operator whose sole purpose is to carry out the program" (Melzak 1961:283). B-B-J (loc. cit.) add the stipulation that the holes are "capable of holding any number of stones" (p. 46). Both Melzak and Lambek appear in *The Canadian Mathematical Bulletin*, vol. 4, no. 3, September 1961.
48. If no confusion results, the word "counters" can be dropped, and a location can be said to contain a single "number".
49. "We say that an instruction is *effective* if there is a procedure that the robot can follow in order to determine precisely how to obey the instruction." (Stone 1972:6)
50. cf Minsky 1967: Chapter 11 "Computer models" and Chapter 14 "Very Simple Bases for Computability" pp. 255–281, in particular,
51. cf Knuth 1973:3.
52. But always preceded by IF-THEN to avoid improper subtraction.
53. Knuth 1973:4
54. Stone 1972:5. Methods for extracting roots are not trivial: see Methods of computing square roots.

55. Leeuwen, Jan (1990). *Handbook of Theoretical Computer Science: Algorithms and complexity. Volume A* (https://books.google.com/books?id=-X39_rA3VSQC). Elsevier. p. 85. ISBN 978-0-444-88071-0.
56. John G. Kemeny and Thomas E. Kurtz 1985 *Back to Basic: The History, Corruption, and Future of the Language*, Addison-Wesley Publishing Company, Inc. Reading, MA, ISBN 0-201-13433-0.
57. Tausworthe 1977:101
58. Tausworthe 1977:142
59. Knuth 1973 section 1.2.1, expanded by Tausworthe 1977 at pages 100ff and Chapter 9.1
60. cf Tausworthe 1977
61. Heath 1908:300; Hawking's Dover 2005 edition derives from Heath.
62. " 'Let CD, measuring BF, leave FA less than itself.' This is a neat abbreviation for saying, measure along BA successive lengths equal to CD until a point F is reached such that the length FA remaining is less than CD; in other words, let BF be the largest exact multiple of CD contained in BA" (Heath 1908:297)
63. For modern treatments using division in the algorithm, see Hardy and Wright 1979:180, Knuth 1973:2 (Volume 1), plus more discussion of Euclid's algorithm in Knuth 1969:293–297 (Volume 2).
64. Euclid covers this question in his Proposition 1.
65. "Euclid's Elements, Book VII, Proposition 2" (http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html). Aleph0.clarku.edu. Archived (https://web.archive.org/web/20120524074919/http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html) from the original on May 24, 2012. Retrieved May 20, 2012.
66. While this notion is in widespread use, it cannot be defined precisely.
67. Knuth 1973:13–18. He credits "the formulation of algorithm-proving in terms of assertions and induction" to R W. Floyd, Peter Naur, C.A.R. Hoare, H.H. Goldstine and J. von Neumann. Tausworth 1977 borrows Knuth's Euclid example and extends Knuth's method in section 9.1 *Formal Proofs* (pp. 288–298).
68. Tausworthe 1997:294
69. cf Knuth 1973:7 (Vol. I), and his more-detailed analyses on pp. 1969:294–313 (Vol II).
70. Breakdown occurs when an algorithm tries to compact itself. Success would solve the Halting problem.
71. Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "The (black) art of run-time evaluation: Are we comparing algorithms or implementations?". *Knowledge and Information Systems*. **52** (2): 341–378. doi:10.1007/s10115-016-1004-2 (https://doi.org/10.1007%2Fs10115-016-1004-2). ISSN 0219-1377 (https://www.worldcat.org/issn/0219-1377). S2CID 40772241 (https://api.semanticscholar.org/CorpusID:40772241).
72. Gillian Conahan (January 2013). "Better Math Makes Faster Data Networks" (http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks). discovermagazine.com. Archived (https://web.archive.org/web/20140513212427/http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks) from the original on May 13, 2014. Retrieved May 13, 2014.
73. Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, "ACM-SIAM Symposium On Discrete Algorithms (SODA) (http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) Archived (https://web.archive.org/web/20130704180806/http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) July 4, 2013, at the Wayback Machine, Kyoto, January 2012. See also the sFFT Web Page (http://groups.csail.mit.edu/netmit/sFFT/) Archived (https://web.archive.org/web/20120221145740/http://groups.csail.mit.edu/netmit/sFFT/) February 21, 2012, at the Wayback Machine.
74. Kowalski 1979

75. Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). *Knapsack Problems | Hans Kellerer | Springer* (https://www.springer.com/us/book/9783540402862). Springer. doi:10.1007/978-3-540-24777-7 (https://doi.org/10.1007%2F978-3-540-24777-7). ISBN 978-3-540-40286-2. S2CID 28836720 (https://api.semanticscholar.org/CorpusID:28836720). Archived (https://web.archive.org/web/20171018181055/https://www.springer.com/us/book/9783540402862) from the original on October 18, 2017. Retrieved September 19, 2017.
76. Carroll, Sue; Daughtrey, Taz (July 4, 2007). *Fundamental Concepts for the Software Quality Engineer* (https://books.google.com/books?id=bz_cl3B05IcC&pg=PA282). American Society for Quality. pp. 282 et seq. ISBN 978-0-87389-720-4.
77. For instance, the volume of a convex polytope (described using a membership oracle) can be approximated to high accuracy by a randomized polynomial time algorithm, but not by a deterministic one: see Dyer, Martin; Frieze, Alan; Kannan, Ravi (January 1991), "A Random Polynomial-time Algorithm for Approximating the Volume of Convex Bodies", *J. ACM*, **38** (1): 1–17, CiteSeerX 10.1.1.145.4600 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.4600), doi:10.1145/102782.102783 (https://doi.org/10.1145%2F102782.102783), S2CID 13268711 (https://api.semanticscholar.org/CorpusID:13268711).
78. George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.
79. Tsypkin (1971). *Adaptation and learning in automatic systems* (https://books.google.com/books?id=sgDHJlafMskC&pg=PA54). Academic Press. p. 54. ISBN 978-0-08-095582-7.
80. Knuth, Donald E. (1972). "Ancient Babylonian Algorithms" (https://web.archive.org/web/20121224100137/http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF). *Commun. ACM*. **15** (7): 671–677. doi:10.1145/361454.361514 (https://doi.org/10.1145%2F361454.361514). ISSN 0001-0782 (https://www.worldcat.org/issn/0001-0782). S2CID 7829945 (https://api.semanticscholar.org/CorpusID:7829945). Archived from the original (http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF) on December 24, 2012.
81. Aaboe, Asger (2001), *Episodes from the Early History of Astronomy*, New York: Springer, pp. 40–62, ISBN 978-0-387-95136-2
82. Ast, Courtney. "Eratosthenes" (http://www.math.wichita.edu/history/men/eratosthenes.html). Wichita State University: Department of Mathematics and Statistics. Archived (https://web.archive.org/web/20150227150653/http://www.math.wichita.edu/history/men/eratosthenes.html) from the original on February 27, 2015. Retrieved February 27, 2015.
83. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*. Springer Science & Business Media. p. 2. ISBN 9783642181924.
84. Davis 2000:18
85. Bolter 1984:24
86. Bolter 1984:26
87. Bolter 1984:33–34, 204–206.
88. All quotes from W. Stanley Jevons 1880 *Elementary Lessons in Logic: Deductive and Inductive*, Macmillan and Co., London and New York. Republished as a googlebook; cf Jevons 1880:199–201. Louis Couturat 1914 *the Algebra of Logic*, The Open Court Publishing Company, Chicago and London. Republished as a googlebook; cf Couturat 1914:75–76 gives a few more details; he compares this to a typewriter as well as a piano. Jevons states that the account is to be found at January 20, 1870 *The Proceedings of the Royal Society*.
89. Jevons 1880:199–200
90. All quotes from John Venn 1881 *Symbolic Logic*, Macmillan and Co., London. Republished as a googlebook. cf Venn 1881:120–125. The interested reader can find a deeper explanation in those pages.
91. Bell and Newell diagram 1971:39, cf. Davis 2000
92. * Melina Hill, Valley News Correspondent, *A Tinkerer Gets a Place in History*, Valley News West Lebanon NH, Thursday, March 31, 1983, p. 13.
93. Davis 2000:14

94. van Heijenoort 1967:81ff
95. van Heijenoort's commentary on Frege's *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought* in van Heijenoort 1967:1
96. Dixon 1906, cf. Kleene 1952:36–40
97. cf. footnote in Alonzo Church 1936a in Davis 1965:90 and 1936b in Davis 1965:110
98. Kleene 1935–6 in Davis 1965:237ff, Kleene 1943 in Davis 1965:255ff
99. Church 1936 in Davis 1965:88ff
100. cf. "Finite Combinatory Processes – formulation 1", Post 1936 in Davis 1965:289–290
101. Turing 1936–37 in Davis 1965:116ff
102. Rosser 1939 in Davis 1965:226
103. Kleene 1943 in Davis 1965:273–274
104. Kleene 1952:300, 317
105. Kleene 1952:376
106. Turing 1936–37 in Davis 1965:289–290
107. Turing 1936 in Davis 1965, Turing 1939 in Davis 1965:160
108. Hodges, p. 96
109. Turing 1936–37:116
110. Turing 1936–37 in Davis 1965:136
111. Turing 1939 in Davis 1965:160

Bibliography

- Axt, P (1959). "On a Subrecursive Hierarchy and Primitive Recursive Degrees" (https://doi.org/10.2307%2F1993169). *Transactions of the American Mathematical Society*. **92** (1): 85–105. doi:10.2307/1993169 (https://doi.org/10.2307%2F1993169). JSTOR 1993169 (https://www.jstor.org/stable/1993169).
- Bell, C. Gordon and Newell, Allen (1971), *Computer Structures: Readings and Examples*, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.
- Blass, Andreas; Gurevich, Yuri (2003). "Algorithms: A Quest for Absolute Definitions" (http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF). *Bulletin of European Association for Theoretical Computer Science*. **81**. Includes an excellent bibliography of 56 references.
- Bolter, David J. (1984). *Turing's Man: Western Culture in the Computer Age* (1984 ed.). Chapel Hill, NC: The University of North Carolina Press. ISBN 978-0-8078-1564-9., ISBN 0-8078-4108-0
- Boolos, George; Jeffrey, Richard (1999) [1974]. *Computability and Logic* (https://archive.org/details/computabilitylog0000bool_r8y9) (4th ed.). Cambridge University Press, London. ISBN 978-0-521-20402-6.: cf. Chapter 3 *Turing machines* where they discuss "certain enumerable sets not effectively (mechanically) enumerable".
- Burgin, Mark (2004). *Super-Recursive Algorithms*. Springer. ISBN 978-0-387-95569-8.
- Campagnolo, M.L., Moore, C., and Costa, J.F. (2000) An analog characterization of the subrecursive functions. In *Proc. of the 4th Conference on Real Numbers and Computers*, Odense University, pp. 91–109
- Church, Alonzo (1936). "An Unsolvable Problem of Elementary Number Theory". *The American Journal of Mathematics*. **58** (2): 345–363. doi:10.2307/2371045 (https://doi.org/10.2307%2F2371045). JSTOR 2371045 (https://www.jstor.org/stable/2371045). Reprinted in *The Undecidable*, p. 89ff. The first expression of "Church's Thesis". See in particular page 100 (*The Undecidable*) where he defines the notion of "effective calculability" in terms of "an algorithm", and he uses the word "terminates", etc.

- Church, Alonzo (1936). "A Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (1): 40–41. doi:10.2307/2269326 (https://doi.org/10.2307%2F2269326). JSTOR 2269326 (https://www.jstor.org/stable/2269326). S2CID 42323521 (https://api.semanticscholar.org/CorpusID:42323521). Church, Alonzo (1936). "Correction to a Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (3): 101–102. doi:10.2307/2269030 (https://doi.org/10.2307%2F2269030). JSTOR 2269030 (https://www.jstor.org/stable/2269030). S2CID 5557237 (https://api.semanticscholar.org/CorpusID:5557237). Reprinted in *The Undecidable*, p. 110ff. Church shows that the Entscheidungsproblem is unsolvable in about 3 pages of text and 3 pages of footnotes.
- Daffa', Ali Abdullah al- (1977). *The Muslim contribution to mathematics*. London: Croom Helm. ISBN 978-0-85664-464-1.
- Davis, Martin (1965). *The Undecidable: Basic Papers On Undecidable Propositions, Unsolvable Problems and Computable Functions* (https://archive.org/details/undecidablebasic0000davi). New York: Raven Press. ISBN 978-0-486-43228-1. Davis gives commentary before each article. Papers of Gödel, Alonzo Church, Turing, Rosser, Kleene, and Emil Post are included; those cited in the article are listed here by author's name.
- Davis, Martin (2000). *Engines of Logic: Mathematicians and the Origin of the Computer*. New York: W.W. Nortion. ISBN 978-0-393-32229-3. Davis offers concise biographies of Leibniz, Boole, Frege, Cantor, Hilbert, Gödel and Turing with von Neumann as the show-stealing villain. Very brief bios of Joseph-Marie Jacquard, Babbage, Ada Lovelace, Claude Shannon, Howard Aiken, etc.
- This article incorporates public domain material from the NIST document: Black, Paul E. "algorithm" (https://xlinux.nist.gov/dads/HTML/algorithm.html). *Dictionary of Algorithms and Data Structures*.
- Dean, Tim (2012). "Evolution and moral diversity" (https://doi.org/10.4148%2Fbiyclc.v7i0.1775). *Baltic International Yearbook of Cognition, Logic and Communication*. **7**. doi:10.4148/biyclc.v7i0.1775 (https://doi.org/10.4148%2Fbiyclc.v7i0.1775).
- Dennett, Daniel (1995). *Darwin's Dangerous Idea* (https://archive.org/details/darwinsdangerous0000denn). *Complexity*. Vol. 2. New York: Touchstone/Simon & Schuster. pp. 32 (https://archive.org/details/darwinsdangerous0000denn/page/32)–36. Bibcode:1996Cmplx...2a..32M (https://ui.adsabs.harvard.edu/abs/1996Cmplx...2a..32M). doi:10.1002/(SICI)1099-0526(199609/10)2:1<32::AID-CPLX8>3.0.CO;2-H (https://doi.org/10.1002%2F%28SICI%2910 99-0526%28199609%2F10%292%3A1%3C32%3A%3AAID-CPLX8%3E3.0.CO%3B2-H). ISBN 978-0-684-80290-9.
- Dilson, Jesse (2007). *The Abacus* (https://archive.org/details/abacusworldsfirs0000dils) ((1968, 1994) ed.). St. Martin's Press, NY. ISBN 978-0-312-10409-2., ISBN 0-312-10409-X
- Yuri Gurevich, *Sequential Abstract State Machines Capture Sequential Algorithms* (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.3017&rep=rep1&type=pdf), ACM Transactions on Computational Logic, Vol 1, no 1 (July 2000), pp. 77–111. Includes bibliography of 33 sources.
- van Heijenoort, Jean (2001). *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931* ((1967) ed.). Harvard University Press, Cambridge. ISBN 978-0-674-32449-7., 3rd edition 1976[?], ISBN 0-674-32449-8 (pbk.)
- Hodges, Andrew (1983). *Alan Turing: The Enigma*. *Physics Today*. Vol. 37. New York: Simon and Schuster. pp. 107–108. Bibcode:1984PhT....37k.107H (https://ui.adsabs.harvard.edu/abs/1984PhT....37k.107H). doi:10.1063/1.2915935 (https://doi.org/10.1063%2F1.2915935). ISBN 978-0-671-49207-6., ISBN 0-671-49207-1. Cf. Chapter "The Spirit of Truth" for a history leading to, and a discussion of, his proof.

- Kleene, Stephen C. (1936). "General Recursive Functions of Natural Numbers" (https://web.archive.org/web/20140903092121/http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1). *Mathematische Annalen*. **112** (5): 727–742. doi:10.1007/BF01565439 (https://doi.org/10.1007%2FBF01565439). S2CID 120517999 (https://api.semanticscholar.org/CorpusID:120517999). Archived from the original (http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1) on September 3, 2014. Retrieved September 30, 2013. Presented to the American Mathematical Society, September 1935. Reprinted in *The Undecidable*, p. 237ff. Kleene's definition of "general recursion" (known now as mu-recursion) was used by Church in his 1935 paper *An Unsolvable Problem of Elementary Number Theory* that proved the "decision problem" to be "undecidable" (i.e., a negative result).
- Kleene, Stephen C. (1943). "Recursive Predicates and Quantifiers" (https://doi.org/10.2307%2F1990131). *Transactions of the American Mathematical Society*. **53** (1): 41–73. doi:10.2307/1990131 (https://doi.org/10.2307%2F1990131). JSTOR 1990131 (https://www.jstor.org/stable/1990131). Reprinted in *The Undecidable*, p. 255ff. Kleene refined his definition of "general recursion" and proceeded in his chapter "12. Algorithmic theories" to posit "Thesis I" (p. 274); he would later repeat this thesis (in Kleene 1952:300) and name it "Church's Thesis" (Kleene 1952:317) (i.e., the Church thesis).
- Kleene, Stephen C. (1991) [1952]. *Introduction to Metamathematics* (Tenth ed.). North-Holland Publishing Company. ISBN 978-0-7204-2103-3.
- Knuth, Donald (1997). *Fundamental Algorithms, Third Edition*. Reading, Massachusetts: Addison–Wesley. ISBN 978-0-201-89683-1.
- Knuth, Donald (1969). *Volume 2/Seminumerical Algorithms, The Art of Computer Programming First Edition*. Reading, Massachusetts: Addison–Wesley.
- Kosovsky, N.K. *Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms*, LSU Publ., Leningrad, 1981
- Kowalski, Robert (1979). "Algorithm=Logic+Control". *Communications of the ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://doi.org/10.1145%2F359131.359136). S2CID 2509896 (https://api.semanticscholar.org/CorpusID:2509896).
- A.A. Markov (1954) *Theory of algorithms*. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954 [i.e., Jerusalem, Israel Program for Scientific Translations, 1961; available from the Office of Technical Services, U.S. Dept. of Commerce, Washington] Description 444 p. 28 cm. Added t.p. in Russian Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algerifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.]
- Minsky, Marvin (1967). *Computation: Finite and Infinite Machines* (https://archive.org/details/computationfinit0000mins) (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 978-0-13-165449-5. Minsky expands his "...idea of an algorithm – an effective procedure..." in chapter 5.1 *Computability, Effective Procedures and Algorithms. Infinite machines.*
- Post, Emil (1936). "Finite Combinatory Processes, Formulation I". *The Journal of Symbolic Logic*. **1** (3): 103–105. doi:10.2307/2269031 (https://doi.org/10.2307%2F2269031). JSTOR 2269031 (https://www.jstor.org/stable/2269031). S2CID 40284503 (https://api.semanticscholar.org/CorpusID:40284503). Reprinted in *The Undecidable*, pp. 289ff. Post defines a simple algorithmic-like process of a man writing marks or erasing marks and going from box to box and eventually halting, as he follows a list of simple instructions. This is cited by Kleene as one source of his "Thesis I", the so-called Church–Turing thesis.
- Rogers, Jr, Hartley (1987). *Theory of Recursive Functions and Effective Computability*. The MIT Press. ISBN 978-0-262-68052-3.

- Rosser, J.B. (1939). "An Informal Exposition of Proofs of Godel's Theorem and Church's Theorem". *Journal of Symbolic Logic*. **4** (2): 53–60. doi:10.2307/2269059 (https://doi.org/10.2307%2F2269059). JSTOR 2269059 (https://www.jstor.org/stable/2269059). S2CID 39499392 (https://api.semanticscholar.org/CorpusID:39499392). Reprinted in *The Undecidable*, p. 223ff. Herein is Rosser's famous definition of "effective method": "...a method each step of which is precisely predetermined and which is certain to produce the answer in a finite number of steps... a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer" (p. 225–226, *The Undecidable*)
- Santos-Lang, Christopher (2014). "Moral Ecology Approaches to Machine Ethics" (http://grinfree.com/MoralEcology.pdf) (PDF). In van Rysewyk, Simon; Pontier, Matthijs (eds.). *Machine Medical Ethics*. Intelligent Systems, Control and Automation: Science and Engineering. Vol. 74. Switzerland: Springer. pp. 111–127. doi:10.1007/978-3-319-08108-3_8 (https://doi.org/10.1007%2F978-3-319-08108-3_8). ISBN 978-3-319-08107-6.
- Scott, Michael L. (2009). *Programming Language Pragmatics* (3rd ed.). Morgan Kaufmann Publishers/Elsevier. ISBN 978-0-12-374514-9.
- Sipser, Michael (2006). *Introduction to the Theory of Computation* (https://archive.org/details/introductiontoth00sips). PWS Publishing Company. ISBN 978-0-534-94728-6.
- Sober, Elliott; Wilson, David Sloan (1998). *Unto Others: The Evolution and Psychology of Unselfish Behavior* (https://archive.org/details/untoothersevolut00sobe). Cambridge: Harvard University Press. ISBN 9780674930469.
- Stone, Harold S. (1972). *Introduction to Computer Organization and Data Structures* (1972 ed.). McGraw-Hill, New York. ISBN 978-0-07-061726-1. Cf. in particular the first chapter titled: *Algorithms, Turing Machines, and Programs*. His succinct informal definition: "...any sequence of instructions that can be obeyed by a robot, is called an *algorithm*" (p. 4).
- Tausworthe, Robert C (1977). *Standardized Development of Computer Software Part 1 Methods*. Englewood Cliffs NJ: Prentice–Hall, Inc. ISBN 978-0-13-842195-3.
- Turing, Alan M. (1936–37). "On Computable Numbers, With An Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. Series 2. **42**: 230–265. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712).. Corrections, ibid, vol. 43(1937) pp. 544–546. Reprinted in *The Undecidable*, p. 116ff. Turing's famous paper completed as a Master's dissertation while at King's College Cambridge UK.
- Turing, Alan M. (1939). "Systems of Logic Based on Ordinals". *Proceedings of the London Mathematical Society*. **45**: 161–228. doi:10.1112/plms/s2-45.1.161 (https://doi.org/10.1112%2Fplms%2Fs2-45.1.161). hdl:21.11116/0000-0001-91CE-3 (https://hdl.handle.net/21.11116%2F0000-0001-91CE-3). Reprinted in *The Undecidable*, pp. 155ff. Turing's paper that defined "the oracle" was his PhD thesis while at Princeton.
- United States Patent and Trademark Office (2006), *2106.02 **>Mathematical Algorithms: 2100 Patentability* (http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2106_02.htm), Manual of Patent Examining Procedure (MPEP). Latest revision August 2006

Further reading

- Bellah, Robert Neelly (1985). *Habits of the Heart: Individualism and Commitment in American Life* (https://books.google.com/books?id=XsUojihVZQcC). Berkeley: University of California Press. ISBN 978-0-520-25419-0.
- Berlinski, David (2001). *The Advent of the Algorithm: The 300-Year Journey from an Idea to the Computer* (https://archive.org/details/adventofalgorith0000berl). Harvest Books. ISBN 978-0-15-601391-8.
- Chabert, Jean-Luc (1999). *A History of Algorithms: From the Pebble to the Microchip*. Springer Verlag. ISBN 978-3-540-63369-3.

- Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2009). *Introduction To Algorithms* (3rd ed.). MIT Press. ISBN 978-0-262-03384-8.
- Harel, David; Feldman, Yishai (2004). *Algorithmics: The Spirit of Computing*. Addison-Wesley. ISBN 978-0-321-11784-7.
- Hertzke, Allen D.; McRorie, Chris (1998). "The Concept of Moral Ecology". In Lawler, Peter Augustine; McConkey, Dale (eds.). *Community and Political Thought Today*. Westport, CT: Praeger.
- Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms (http://www-cs-faculty.stanford.edu/~uno/aa.html)*. Stanford, California: Center for the Study of Language and Information.
- Knuth, Donald E. (2010). *Selected Papers on Design of Algorithms (http://www-cs-faculty.stanford.edu/~uno/da.html)*. Stanford, California: Center for the Study of Language and Information.
- Wallach, Wendell; Allen, Colin (November 2008). *Moral Machines: Teaching Robots Right from Wrong*. US: Oxford University Press. ISBN 978-0-19-537404-9.
- Bleakley, Chris (2020). *Poems that Solve Puzzles: The History and Science of Algorithms* (https://books.google.com/books?id=3pr5DwAAQBAJ). Oxford University Press. ISBN 978-0-19-885373-2.

External links

- "Algorithm" (https://www.encyclopediaofmath.org/index.php?title=Algorithm), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Algorithms (https://curlie.org/Computers/Algorithms/) at Curlie
- Weisstein, Eric W. "Algorithm" (https://mathworld.wolfram.com/Algorithm.html). *MathWorld*.
- Dictionary of Algorithms and Data Structures (https://www.nist.gov/dads/) – National Institute of Standards and Technology

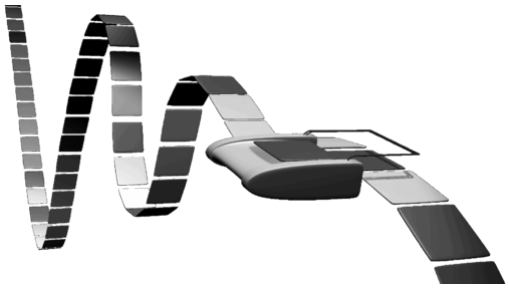
Algorithm repositories

- The Stony Brook Algorithm Repository (http://www.cs.sunysb.edu/~algorith/) – State University of New York at Stony Brook
- Collected Algorithms of the ACM (http://calgo.acm.org/) – Association for Computing Machinery
- The Stanford GraphBase (http://www-cs-staff.stanford.edu/~knuth/sgb.html) – Stanford University

---

Theory of computation

In theoretical computer science and mathematics, the **theory of computation** is the branch that deals with what problems can be solved on a model of computation, using an algorithm, how efficiently they can be solved or to what degree (e.g., approximate solutions versus precise ones). The field is divided into three major branches: automata theory and formal languages, computability theory, and computational complexity theory, which are linked by the question: *"What are the fundamental capabilities and limitations of computers?"*.[1]



An artistic representation of a Turing machine. Turing machines are frequently used as theoretical models for computing.

In order to perform a rigorous study of computation, computer scientists work with a mathematical abstraction of computers called a model of computation. There are several models in use, but the most commonly examined is the Turing machine.[2] Computer scientists study the Turing machine because it is simple to formulate, can be analyzed and used to prove results, and because it represents what many consider the most powerful possible "reasonable" model of computation (see Church–Turing thesis).[3] It might seem that the potentially infinite memory capacity is an unrealizable attribute, but any decidable problem[4] solved by a Turing machine will always require only a finite amount of memory. So in principle, any problem that can be solved (decided) by a Turing machine can be solved by a computer that has a finite amount of memory.

Contents

History

The theory of computation can be considered the creation of models of all kinds in the field of computer science. Therefore, mathematics and logic are used. In the last century it became an independent academic discipline and was separated from mathematics.

Some pioneers of the theory of computation were Ramon Llull, Alonzo Church, Kurt Gödel, Alan Turing, Stephen Kleene, Rózsa Péter, John von Neumann and Claude Shannon.

Branches

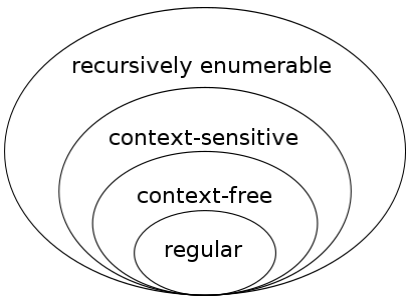
Automata theory

Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \to \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \to \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \to \gamma$
Type-3	Regular	Finite state automaton	$A \to a$ and $A \to aB$

Automata theory is the study of abstract machines (or more appropriately, abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata. Automata comes from the Greek word (Αυτόματα) which means that something is doing something by itself. Automata theory is also closely related to formal language theory,[5] as the automata are often classified by the class of formal languages they are able to recognize. An automaton can be a finite representation of a formal language that may be an infinite set. Automata are used as theoretical models for computing machines, and are used for proofs about computability.

Formal Language theory

Language theory is a branch of mathematics concerned with describing languages as a set of operations over an alphabet. It is closely linked with automata theory, as automata are used to generate and recognize formal languages. There are several classes of formal languages, each allowing more complex language specification than the one before it, i.e. Chomsky hierarchy,[6] and each corresponding to a class of automata which recognizes it. Because automata are used as models for computation, formal languages are the preferred mode of specification for any problem that must be computed.



Set inclusions described by the Chomsky hierarchy

Computability theory

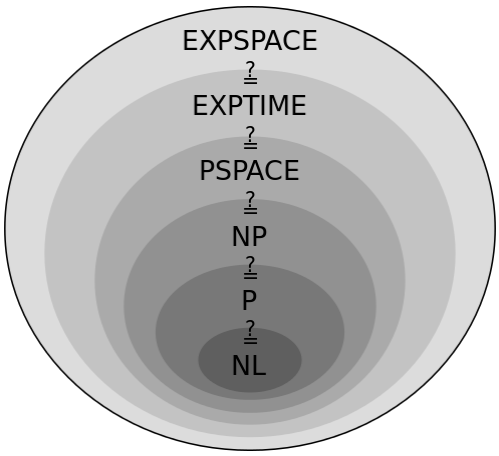
Computability theory deals primarily with the question of the extent to which a problem is solvable on a computer. The statement that the halting problem cannot be solved by a Turing machine[7] is one of the most important results in computability theory, as it is an example of a concrete problem that is both easy to formulate and impossible to solve using a Turing machine. Much of computability theory builds on the halting problem result.

Another important step in computability theory was Rice's theorem, which states that for all non-trivial properties of partial functions, it is undecidable whether a Turing machine computes a partial function with that property.[8]

Computability theory is closely related to the branch of mathematical logic called recursion theory, which removes the restriction of studying only models of computation which are reducible to the Turing model.[9] Many mathematicians and computational theorists who study recursion theory will refer to it as computability theory.

Computational complexity theory

Complexity theory considers not only whether a problem can be solved at all on a computer, but also how efficiently the problem can be solved. Two major aspects are considered: time complexity and space complexity, which are respectively how many steps does it take to perform a computation, and how much memory is required to perform that computation.



A representation of the relation among complexity classes

In order to analyze how much time and space a given algorithm requires, computer scientists express the time or space required to solve the problem as a function of the size of the input problem. For example, finding a particular number in a long list of numbers becomes harder as the list of numbers grows larger. If we say there are n numbers in the list, then if the list is not sorted or indexed in any way we may have to look at every number in order to find the number we're seeking. We thus say that in order to solve this problem, the computer needs to perform a number of steps that grows linearly in the size of the problem.

To simplify this problem, computer scientists have adopted Big O notation, which allows functions to be compared in a way that ensures that particular aspects of a machine's construction do not need to be considered, but rather only the asymptotic behavior as problems become large. So in our previous example, we might say that the problem requires $O(n)$ steps to solve.

Perhaps the most important open problem in all of computer science is the question of whether a certain broad class of problems denoted NP can be solved efficiently. This is discussed further at Complexity classes P and NP, and P versus NP problem is one of the seven Millennium Prize Problems stated by the Clay Mathematics Institute in 2000. The Official Problem Description (http://www.claymath.org/sites/default/files/pvsnp.pdf) was given by Turing Award winner Stephen Cook.

Models of computation

Aside from a Turing machine, other equivalent (See: Church–Turing thesis) models of computation are in use.

Lambda calculus
: A computation consists of an initial lambda expression (or two if you want to separate the function and its input) plus a finite sequence of lambda terms, each deduced from the preceding term by one application of Beta reduction.

Combinatory logic
: is a concept which has many similarities to λ-calculus, but also important differences exist (e.g. fixed point combinator **Y** has normal form in combinatory logic but not in λ-calculus). Combinatory logic was developed with great ambitions: understanding the nature of paradoxes, making foundations of mathematics more economic (conceptually), eliminating the notion of variables (thus clarifying their role in mathematics).

μ-recursive functions

a computation consists of a mu-recursive function, *i.e.* its defining sequence, any input value(s) and a sequence of recursive functions appearing in the defining sequence with inputs and outputs. Thus, if in the defining sequence of a recursive function $f(x)$ the functions $g(x)$ and $h(x, y)$ appear, then terms of the form 'g(5)=7' or 'h(3,2)=10' might appear. Each entry in this sequence needs to be an application of a basic function or follow from the entries above by using composition, primitive recursion or μ recursion. For instance if $f(x) = h(x, g(x))$, then for 'f(5)=3' to appear, terms like 'g(5)=6' and 'h(5,6)=3' must occur above. The computation terminates only if the final term gives the value of the recursive function applied to the inputs.

Markov algorithm
: a string rewriting system that uses grammar-like rules to operate on strings of symbols.

Register machine
: is a theoretically interesting idealization of a computer. There are several variants. In most of them, each register can hold a natural number (of unlimited size), and the instructions are simple (and few in number), e.g. only decrementation (combined with conditional jump) and incrementation exist (and halting). The lack of the infinite (or dynamically growing) external store (seen at Turing machines) can be understood by replacing its role with Gödel numbering techniques: the fact that each register holds a natural number allows the possibility of representing a complicated thing (e.g. a sequence, or a matrix etc.) by an appropriate huge natural number — unambiguity of both representation and interpretation can be established by number theoretical foundations of these techniques.

In addition to the general computational models, some simpler computational models are useful for special, restricted applications. Regular expressions, for example, specify string patterns in many contexts, from office productivity software to programming languages. Another formalism mathematically equivalent to regular expressions, Finite automata are used in circuit design and in some kinds of problem-solving. Context-free grammars specify programming language syntax. Non-deterministic pushdown automata are another formalism equivalent to context-free grammars. Primitive recursive functions are a defined subclass of the recursive functions.

Different models of computation have the ability to do different tasks. One way to measure the power of a computational model is to study the class of formal languages that the model can generate; in such a way to the Chomsky hierarchy of languages is obtained.

References

1. Michael Sipser (2013). *Introduction to the Theory of Computation 3rd*. Cengage Learning. ISBN 978-1-133-18779-0. "central areas of the theory of computation: automata, computability, and complexity. (Page 1)"
2. Hodges, Andrew (2012). *Alan Turing: The Enigma* (The Centenary ed.). Princeton University Press. ISBN 978-0-691-15564-7.
3. Rabin, Michael O. (June 2012). *Turing, Church, Gödel, Computability, Complexity and Randomization: A Personal View* (http://videolectures.net/turing100_rabin_turing_church_goedel/).
4. Donald Monk (1976). *Mathematical Logic* (https://archive.org/details/mathematicallogi00jdon). Springer-Verlag. ISBN 9780387901701.
5. Hopcroft, John E. and Jeffrey D. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation. 3rd ed*. Reading, MA: Addison-Wesley. ISBN 978-0-321-45536-9.
6. Chomsky hierarchy (1956). "Three models for the description of language". *Information Theory, IRE Transactions on*. IEEE. **2** (3): 113–124. doi:10.1109/TIT.1956.1056813 (https://doi.org/10.1109%2FTIT.1956.1056813).
7. Alan Turing (1937). "On computable numbers, with an application to the Entscheidungsproblem" (http://www.turingarchive.org/browse.php/B/12). *Proceedings of the London Mathematical Society*. IEEE. **2** (42): 230–265. doi:10.1112/plms/s2-42.1.230 (https://do

i.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712). Retrieved 6 January 2015.
8. Henry Gordon Rice (1953). "Classes of Recursively Enumerable Sets and Their Decision Problems" (https://doi.org/10.2307%2F1990888). *Transactions of the American Mathematical Society*. American Mathematical Society. **74** (2): 358–366. doi:10.2307/1990888 (https://doi.org/10.2307%2F1990888). JSTOR 1990888 (https://www.jstor.org/stable/1990888).
9. Martin Davis (2004). *The undecidable: Basic papers on undecidable propositions, unsolvable problems and computable functions (Dover Ed)*. Dover Publications. ISBN 978-0486432281.

Further reading

Textbooks aimed at computer scientists

(There are many textbooks in this area; this list is by necessity incomplete.)

- Hopcroft, John E., and Jeffrey D. Ullman (2006). *Introduction to Automata Theory, Languages, and Computation. 3rd ed* Reading, MA: Addison-Wesley. ISBN 978-0-321-45536-9 One of the standard references in the field.
- Linz P (2007). *An introduction to formal language and automata*. Narosa Publishing. ISBN 9788173197819.
- Michael Sipser (2013). *Introduction to the Theory of Computation* (3rd ed.). Cengage Learning. ISBN 978-1-133-18779-0.
- Eitan Gurari (1989). *An Introduction to the Theory of Computation* (https://web.archive.org/web/20070107040625/http://www.cse.ohio-state.edu/~gurari/theory-bk/theory-bk.html). Computer Science Press. ISBN 0-7167-8182-4. Archived from the original (http://www.cse.ohio-state.edu/~gurari/theory-bk/theory-bk.html) on 2007-01-07.
- Hein, James L. (1996) *Theory of Computation.* Sudbury, MA: Jones & Bartlett. ISBN 978-0-86720-497-1 A gentle introduction to the field, appropriate for second-year undergraduate computer science students.
- Taylor, R. Gregory (1998). *Models of Computation and Formal Languages.* New York: Oxford University Press. ISBN 978-0-19-510983-2 An unusually readable textbook, appropriate for upper-level undergraduates or beginning graduate students.
- Lewis, F. D. (2007). *Essentials of theoretical computer science (http://cse.ucdenver.edu/~cscialtman/foundation/Essentials%20of%20Theoretical%20Computer%20Science.pdf)* A textbook covering the topics of formal languages, automata and grammars. The emphasis appears to be on presenting an overview of the results and their applications rather than providing proofs of the results.
- Martin Davis, Ron Sigal, Elaine J. Weyuker, *Computability, complexity, and languages: fundamentals of theoretical computer science*, 2nd ed., Academic Press, 1994, ISBN 0-12-206382-1. Covers a wider range of topics than most other introductory books, including program semantics and quantification theory. Aimed at graduate students.

Books on computability theory from the (wider) mathematical perspective

- Hartley Rogers, Jr (1987). *Theory of Recursive Functions and Effective Computability*, MIT Press. ISBN 0-262-68052-1
- S. Barry Cooper (2004). *Computability Theory*. Chapman and Hall/CRC. ISBN 1-58488-237-9..
- Carl H. Smith, *A recursive introduction to the theory of computation*, Springer, 1994, ISBN 0-387-94332-3. A shorter textbook suitable for graduate students in Computer Science.

Historical perspective

- Richard L. Epstein and Walter A. Carnielli (2000). *Computability: Computable Functions, Logic, and the Foundations of Mathematics, with Computability: A Timeline (2nd ed.)*.

Wadsworth/Thomson Learning. ISBN 0-534-54644-7..

External links

- Theory of Computation (http://toc.csail.mit.edu/) at MIT
- Theory of Computation (http://toc.seas.harvard.edu/) at Harvard
- Computability Logic (http://www.cis.upenn.edu/~giorgi/cl.html) - A theory of interactive computation. The main web source on this subject.

Information theory

Information theory is the scientific study of the quantification, storage, and communication of digital information.[1] The field was fundamentally established by the works of Harry Nyquist and Ralph Hartley, in the 1920s, and Claude Shannon in the 1940s.[2]:vii The field is at the intersection of probability theory, statistics, computer science, statistical mechanics, information engineering, and electrical engineering.

A key measure in information theory is entropy. Entropy quantifies the amount of uncertainty involved in the value of a random variable or the outcome of a random process. For example, identifying the outcome of a fair coin flip (with two equally likely outcomes) provides less information (lower entropy) than specifying the outcome from a roll of a die (with six equally likely outcomes). Some other important measures in information theory are mutual information, channel capacity, error exponents, and relative entropy. Important sub-fields of information theory include source coding, algorithmic complexity theory, algorithmic information theory and information-theoretic security.

Applications of fundamental topics of information theory include source coding/data compression (e.g. for ZIP files), and channel coding/error detection and correction (e.g. for DSL). Its impact has been crucial to the success of the Voyager missions to deep space, the invention of the compact disc, the feasibility of mobile phones and the development of the Internet. The theory has also found applications in other areas, including statistical inference,[3] cryptography, neurobiology,[4] perception,[5] linguistics, the evolution[6] and function[7] of molecular codes (bioinformatics), thermal physics,[8] molecular dynamics,[9] quantum computing, black holes, information retrieval, intelligence gathering, plagiarism detection,[10] pattern recognition, anomaly detection[11] and even art creation.

Contents

Overview

Information theory studies the transmission, processing, extraction, and utilization of information. Abstractly, information can be thought of as the resolution of uncertainty. In the case of communication of information over a noisy channel, this abstract concept was formalized in 1948 by Claude Shannon in a paper entitled *A Mathematical Theory of Communication*, in which information is thought of as a set of possible messages, and the goal is to send these messages over a noisy channel, and to have the receiver reconstruct the message with low probability of error, in spite of the channel noise. Shannon's main result, the noisy-channel coding theorem showed that, in the limit of many channel uses, the rate of information that is asymptotically achievable is equal to the channel capacity, a quantity dependent merely on the statistics of the channel over which the messages are sent.[4]

Coding theory is concerned with finding explicit methods, called *codes*, for increasing the efficiency and reducing the error rate of data communication over noisy channels to near the channel capacity. These codes can be roughly subdivided into data compression (source coding) and error-correction (channel coding) techniques. In the latter case, it took many years to find the methods Shannon's work proved were possible.

A third class of information theory codes are cryptographic algorithms (both codes and ciphers). Concepts, methods and results from coding theory and information theory are widely used in cryptography and cryptanalysis. *See the article ban (unit) for a historical application.*

Historical background

The landmark event *establishing* the discipline of information theory and bringing it to immediate worldwide attention was the publication of Claude E. Shannon's classic paper "A Mathematical Theory of Communication" in the *Bell System Technical Journal* in July and October 1948.

Prior to this paper, limited information-theoretic ideas had been developed at Bell Labs, all implicitly assuming events of equal probability. Harry Nyquist's 1924 paper, *Certain Factors Affecting Telegraph Speed*, contains a theoretical section quantifying "intelligence" and the "line speed" at which it can be transmitted by a communication system, giving the relation $W = K \log m$ (recalling the Boltzmann constant), where W is the speed of transmission of intelligence, m is the number of different voltage levels to choose from at each time step, and K is a constant. Ralph Hartley's 1928 paper, *Transmission of Information*, uses the word *information* as a measurable quantity, reflecting the receiver's ability to distinguish one sequence of symbols from any other, thus quantifying information as $H = \log S^n = n \log S$, where S was the number of possible symbols, and n the number of symbols in a transmission. The unit of information was therefore the decimal digit, which since has sometimes been called the hartley in his honor as a unit or scale or measure of information. Alan Turing in 1940 used similar ideas as part of the statistical analysis of the breaking of the German second world war Enigma ciphers.

Much of the mathematics behind information theory with events of different probabilities were developed for the field of thermodynamics by Ludwig Boltzmann and J. Willard Gibbs. Connections between information-theoretic entropy and thermodynamic entropy, including the important contributions by Rolf Landauer in the 1960s, are explored in *Entropy in thermodynamics and information theory*.

In Shannon's revolutionary and groundbreaking paper, the work for which had been substantially completed at Bell Labs by the end of 1944, Shannon for the first time introduced the qualitative and quantitative model of communication as a statistical process underlying information theory, opening with the assertion:

> "The fundamental problem of communication is that of reproducing at one point, either exactly or approximately, a message selected at another point."

With it came the ideas of

- the information entropy and redundancy of a source, and its relevance through the source coding theorem;
- the mutual information, and the channel capacity of a noisy channel, including the promise of perfect loss-free communication given by the noisy-channel coding theorem;
- the practical result of the Shannon–Hartley law for the channel capacity of a Gaussian channel; as well as
- the bit—a new way of seeing the most fundamental unit of information.

Quantities of information

Information theory is based on probability theory and statistics, where quantified information is usually described in terms of bits. Information theory often concerns itself with measures of information of the distributions associated with random variables. One of the most important measures is called entropy, which forms the building block of many other measures. Entropy allows quantification of measure of information in a single random variable. Another useful concept is mutual information defined on two random variables, which describes the measure of information in common between those variables, which can be used to describe their correlation. The former quantity is a property of the probability distribution of a random variable and gives a limit on the rate at which data generated by independent samples with the given distribution can be reliably compressed. The latter is a property of the joint distribution of two random variables, and is the maximum rate of reliable communication across a noisy channel in the limit of long block lengths, when the channel statistics are determined by the joint distribution.

The choice of logarithmic base in the following formulae determines the unit of information entropy that is used. A common unit of information is the bit, based on the binary logarithm. Other units include the nat, which is based on the natural logarithm, and the decimal digit, which is based on the common logarithm.

In what follows, an expression of the form $p \log p$ is considered by convention to be equal to zero whenever $p = 0$. This is justified because $\lim_{p \to 0+} p \log p = 0$ for any logarithmic base.

Entropy of an information source

Based on the probability mass function of each source symbol to be communicated, the Shannon entropy H, in units of bits (per symbol), is given by

$$H = -\sum_i p_i \log_2 (p_i)$$

where p_i is the probability of occurrence of the i-th possible value of the source symbol. This equation gives the entropy in the units of "bits" (per symbol) because it uses a logarithm of base 2, and this base-2 measure of entropy has sometimes been called the shannon in his honor. Entropy is also commonly computed using the natural logarithm (base e, where e is Euler's number), which produces a measurement of entropy in nats per symbol and sometimes simplifies the analysis by avoiding the need to include extra constants in the formulas. Other bases are also possible, but less commonly used. For example, a logarithm of base $2^8 = 256$ will produce a measurement in bytes per symbol, and a logarithm of base 10 will produce a measurement in decimal digits (or hartleys) per symbol.

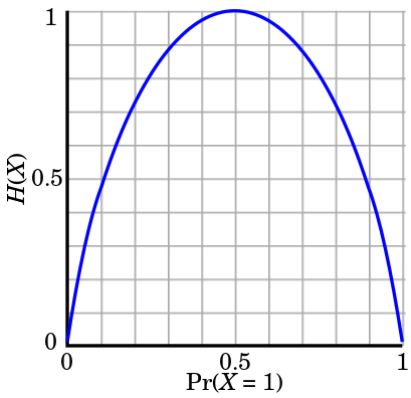
Intuitively, the entropy H_X of a discrete random variable X is a measure of the amount of *uncertainty* associated with the value of X when only its distribution is known.

The entropy of a source that emits a sequence of N symbols that are independent and identically distributed (iid) is $N \cdot H$ bits (per message of N symbols). If the source data symbols are identically distributed but not independent, the entropy of a message of length N will be less than $N \cdot H$.

If one transmits 1000 bits (0s and 1s), and the value of each of these bits is known to the receiver (has a specific value with certainty) ahead of transmission, it is clear that no information is transmitted. If, however, each bit is independently equally likely to be 0 or 1, 1000 shannons of information (more often called bits) have been transmitted. Between these two extremes, information can be quantified as follows. If $\mathbb{X}$ is the set of all messages $\{x_1, \ldots, x_n\}$ that X could be, and $p(x)$ is the probability of some $x \in \mathbb{X}$, then the entropy, H, of X is defined:[12]

$$H(X) = \mathbb{E}_X[I(x)] = -\sum_{x \in \mathbb{X}} p(x) \log p(x).$$

(Here, $I(x)$ is the self-information, which is the entropy contribution of an individual message, and $\mathbb{E}_X$ is the expected value.) A property of entropy is that it is maximized when all the messages in the message space are equiprobable $p(x) = 1/n$; i.e., most unpredictable, in which case $H(X) = \log n$.



The entropy of a Bernoulli trial as a function of success probability, often called the *binary entropy function*, $H_b(p)$. The entropy is maximized at 1 bit per trial when the two possible outcomes are equally probable, as in an unbiased coin toss.

The special case of information entropy for a random variable with two outcomes is the binary entropy function, usually taken to the logarithmic base 2, thus having the shannon (Sh) as unit:

$$H_{\mathrm{b}}(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

Joint entropy

The *joint entropy* of two discrete random variables X and Y is merely the entropy of their pairing: (X, Y). This implies that if X and Y are independent, then their joint entropy is the sum of their individual entropies.

For example, if (X, Y) represents the position of a chess piece—X the row and Y the column, then the joint entropy of the row of the piece and the column of the piece will be the entropy of the position of the piece.

$$H(X, Y) = \mathbb{E}_{X,Y}[-\log p(x, y)] = -\sum_{x,y} p(x, y) \log p(x, y)$$

Despite similar notation, joint entropy should not be confused with *cross entropy*.

Conditional entropy (equivocation)

The *conditional entropy* or *conditional uncertainty* of X given random variable Y (also called the *equivocation* of X about Y) is the average conditional entropy over Y:[13]

$$H(X|Y) = \mathbb{E}_Y[H(X|y)] = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y) = -\sum_{x,y} p(x, y) \log p(x|y).$$

Because entropy can be conditioned on a random variable or on that random variable being a certain value, care should be taken not to confuse these two definitions of conditional entropy, the former of which is in more common use. A basic property of this form of conditional entropy is that:

$$H(X|Y) = H(X, Y) - H(Y).$$

Mutual information (transinformation)

Mutual information measures the amount of information that can be obtained about one random variable by observing another. It is important in communication where it can be used to maximize the amount of information shared between sent and received signals. The mutual information of X relative to Y is given by:

$$I(X;Y) = \mathbb{E}_{X,Y}[SI(x, y)] = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)\, p(y)}$$

where SI (*S*pecific mutual *I*nformation) is the pointwise mutual information.

A basic property of the mutual information is that

$$I(X;Y) = H(X) - H(X|Y).$$

That is, knowing Y, we can save an average of $I(X;\ Y)$ bits in encoding X compared to not knowing Y.

Mutual information is symmetric:

$$I(X;Y) = I(Y;X) = H(X) + H(Y) - H(X,Y).$$

Mutual information can be expressed as the average Kullback–Leibler divergence (information gain) between the posterior probability distribution of X given the value of Y and the prior distribution on X:

$$I(X;Y) = \mathbb{E}_{p(y)}\left[D_{\mathrm{KL}}\left(p(X|Y=y)\|p(X)\right)\right].$$

In other words, this is a measure of how much, on the average, the probability distribution on X will change if we are given the value of Y. This is often recalculated as the divergence from the product of the marginal distributions to the actual joint distribution:

$$I(X;Y) = D_{\mathrm{KL}}\left(p(X,Y)\|p(X)p(Y)\right).$$

Mutual information is closely related to the log-likelihood ratio test in the context of contingency tables and the multinomial distribution and to Pearson's χ^2 test: mutual information can be considered a statistic for assessing independence between a pair of variables, and has a well-specified asymptotic distribution.

Kullback–Leibler divergence (information gain)

The *Kullback–Leibler divergence* (or *information divergence*, *information gain*, or *relative entropy*) is a way of comparing two distributions: a "true" probability distribution $p(X)$, and an arbitrary probability distribution $q(X)$. If we compress data in a manner that assumes $q(X)$ is the distribution underlying some data, when, in reality, $p(X)$ is the correct distribution, the Kullback–Leibler divergence is the number of average additional bits per datum necessary for compression. It is thus defined

$$D_{\mathrm{KL}}\left(p(X)\|q(X)\right) = \sum_{x\in X} -p(x)\log q(x) \ - \sum_{x\in X} -p(x)\log p(x) = \sum_{x\in X} p(x)\log\frac{p(x)}{q(x)}.$$

Although it is sometimes used as a 'distance metric', KL divergence is not a true metric since it is not symmetric and does not satisfy the triangle inequality (making it a semi-quasimetric).

Another interpretation of the KL divergence is the "unnecessary surprise" introduced by a prior from the truth: suppose a number X is about to be drawn randomly from a discrete set with probability distribution $p(x)$. If Alice knows the true distribution $p(x)$, while Bob believes (has a prior) that the distribution is $q(x)$, then Bob will be more surprised than Alice, on average, upon seeing the value of X. The KL divergence is the (objective) expected value of Bob's (subjective) surprisal minus Alice's surprisal, measured in bits if the *log* is in base 2. In this way, the extent to which Bob's prior is "wrong" can be quantified in terms of how "unnecessarily surprised" it is expected to make him.

Other quantities

Other important information theoretic quantities include Rényi entropy (a generalization of entropy), differential entropy (a generalization of quantities of information to continuous distributions), and the conditional mutual information.

Coding theory

A picture showing scratches on the readable surface of a CD-R. Music and data CDs are coded using error correcting codes and thus can still be read even if they have minor scratches using error detection and correction.

Coding theory is one of the most important and direct applications of information theory. It can be subdivided into source coding theory and channel coding theory. Using a statistical description for data, information theory quantifies the number of bits needed to describe the data, which is the information entropy of the source.

- Data compression (source coding): There are two formulations for the compression problem:
  - lossless data compression: the data must be reconstructed exactly;
  - lossy data compression: allocates bits needed to reconstruct the data, within a specified fidelity level measured by a distortion function. This subset of information theory is called *rate–distortion theory*.
- Error-correcting codes (channel coding): While data compression removes as much redundancy as possible, an error-correcting code adds just the right kind of redundancy (i.e., error correction) needed to transmit the data efficiently and faithfully across a noisy channel.

This division of coding theory into compression and transmission is justified by the information transmission theorems, or source–channel separation theorems that justify the use of bits as the universal currency for information in many contexts. However, these theorems only hold in the situation where one transmitting user wishes to communicate to one receiving user. In scenarios with more than one transmitter (the multiple-access channel), more than one receiver (the broadcast channel) or intermediary "helpers" (the relay channel), or more general networks, compression followed by transmission may no longer be optimal.

Source theory

Any process that generates successive messages can be considered a *source* of information. A memoryless source is one in which each message is an independent identically distributed random variable, whereas the properties of ergodicity and stationarity impose less restrictive constraints. All such sources are stochastic. These terms are well studied in their own right outside information theory.

Rate

Information *rate* is the average entropy per symbol. For memoryless sources, this is merely the entropy of each symbol, while, in the case of a stationary stochastic process, it is

$$r = \lim_{n \to \infty} H(X_n | X_{n-1}, X_{n-2}, X_{n-3}, \ldots);$$

that is, the conditional entropy of a symbol given all the previous symbols generated. For the more general case of a process that is not necessarily stationary, the *average rate* is

$$r = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \dots X_n);$$

that is, the limit of the joint entropy per symbol. For stationary sources, these two expressions give the same result.[14]

Information rate is defined as

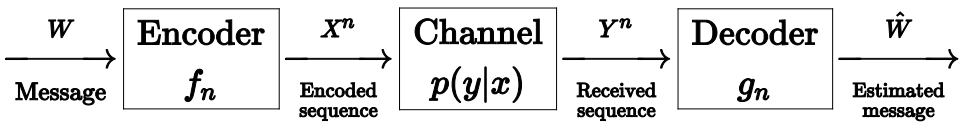
$$r = \lim_{n \to \infty} \frac{1}{n} I(X_1, X_2, \dots X_n; Y_1, Y_2, \dots Y_n);$$

It is common in information theory to speak of the "rate" or "entropy" of a language. This is appropriate, for example, when the source of information is English prose. The rate of a source of information is related to its redundancy and how well it can be compressed, the subject of *source coding*.

Channel capacity

Communications over a channel is the primary motivation of information theory. However, channels often fail to produce exact reconstruction of a signal; noise, periods of silence, and other forms of signal corruption often degrade quality.

Consider the communications process over a discrete channel. A simple model of the process is shown below:



Here X represents the space of messages transmitted, and Y the space of messages received during a unit time over our channel. Let $p(y|x)$ be the conditional probability distribution function of Y given X. We will consider $p(y|x)$ to be an inherent fixed property of our communications channel (representing the nature of the *noise* of our channel). Then the joint distribution of X and Y is completely determined by our channel and by our choice of $f(x)$, the marginal distribution of messages we choose to send over the channel. Under these constraints, we would like to maximize the rate of information, or the *signal*, we can communicate over the channel. The appropriate measure for this is the mutual information, and this maximum mutual information is called the *channel capacity* and is given by:

$$C = \max_f I(X;Y).$$

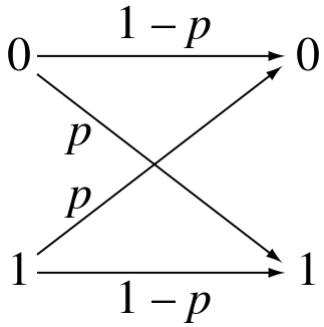
This capacity has the following property related to communicating at information rate R (where R is usually bits per symbol). For any information rate $R < C$ and coding error $\varepsilon > 0$, for large enough N, there exists a code of length N and rate ≥ R and a decoding algorithm, such that the maximal probability of block error is ≤ ε; that is, it is always possible to transmit with arbitrarily small block error. In addition, for any rate $R > C$, it is impossible to transmit with arbitrarily small block error.

Channel coding is concerned with finding such nearly optimal codes that can be used to transmit data over a noisy channel with a small coding error at a rate near the channel capacity.

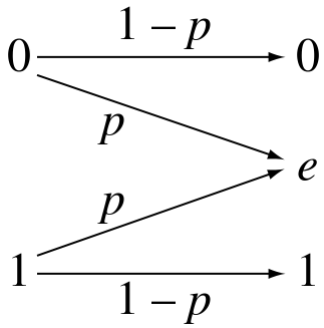
Capacity of particular channel models

- A continuous-time analog communications channel subject to Gaussian noise—see Shannon–Hartley theorem.

- A binary symmetric channel (BSC) with crossover probability *p* is a binary input, binary output channel that flips the input bit with probability *p*. The BSC has a capacity of $1 - H_\mathrm{b}(p)$ bits per channel use, where H_b is the binary entropy function to the base-2 logarithm:



- A binary erasure channel (BEC) with erasure probability *p* is a binary input, ternary output channel. The possible channel outputs are 0, 1, and a third symbol 'e' called an erasure. The erasure represents complete loss of information about an input bit. The capacity of the BEC is 1 – *p* bits per channel use.



Channels with memory and directed information

In practice many channels have memory. Namely, at time i the channel is given by the conditional probability $P(y_i|x_i, x_{i-1}, x_{i-2}, \ldots, x_1, y_{i-1}, y_{i-2}, \ldots, y_1)$.. It is often more comfortable to use the notation $x^i = (x_i, x_{i-1}, x_{i-2}, \ldots, x_1)$ and the channel become $P(y_i|x^i, y^{i-1})$.. In such a case the capacity is given by the mutual information rate when there is no feedback available and the Directed information rate in the case that either there is feedback or not[15][16] (if there is no feedback the directed information equals the mutual information).

Applications to other fields

Intelligence uses and secrecy applications

Information theoretic concepts apply to cryptography and cryptanalysis. Turing's information unit, the ban, was used in the Ultra project, breaking the German Enigma machine code and hastening the end of World War II in Europe. Shannon himself defined an important concept now called the unicity distance. Based on the redundancy of the plaintext, it attempts to give a minimum amount of ciphertext necessary to ensure unique decipherability.

Information theory leads us to believe it is much more difficult to keep secrets than it might first appear. A brute force attack can break systems based on asymmetric key algorithms or on most commonly used methods of symmetric key algorithms (sometimes called secret key algorithms),

such as block ciphers. The security of all such methods currently comes from the assumption that no known attack can break them in a practical amount of time.

Information theoretic security refers to methods such as the one-time pad that are not vulnerable to such brute force attacks. In such cases, the positive conditional mutual information between the plaintext and ciphertext (conditioned on the key) can ensure proper transmission, while the unconditional mutual information between the plaintext and ciphertext remains zero, resulting in absolutely secure communications. In other words, an eavesdropper would not be able to improve his or her guess of the plaintext by gaining knowledge of the ciphertext but not of the key. However, as in any other cryptographic system, care must be used to correctly apply even information-theoretically secure methods; the Venona project was able to crack the one-time pads of the Soviet Union due to their improper reuse of key material.

Pseudorandom number generation

Pseudorandom number generators are widely available in computer language libraries and application programs. They are, almost universally, unsuited to cryptographic use as they do not evade the deterministic nature of modern computer equipment and software. A class of improved random number generators is termed cryptographically secure pseudorandom number generators, but even they require random seeds external to the software to work as intended. These can be obtained via extractors, if done carefully. The measure of sufficient randomness in extractors is min-entropy, a value related to Shannon entropy through Rényi entropy; Rényi entropy is also used in evaluating randomness in cryptographic systems. Although related, the distinctions among these measures mean that a random variable with high Shannon entropy is not necessarily satisfactory for use in an extractor and so for cryptography uses.

Seismic exploration

One early commercial application of information theory was in the field of seismic oil exploration. Work in this field made it possible to strip off and separate the unwanted noise from the desired seismic signal. Information theory and digital signal processing offer a major improvement of resolution and image clarity over previous analog methods.[17]

Semiotics

Semioticians Doede Nauta and Winfried Nöth both considered Charles Sanders Peirce as having created a theory of information in his works on semiotics.[18]:171[19]:137 Nauta defined semiotic information theory as the study of "the internal processes of coding, filtering, and information processing."[18]:91

Concepts from information theory such as redundancy and code control have been used by semioticians such as Umberto Eco and Ferruccio Rossi-Landi to explain ideology as a form of message transmission whereby a dominant social class emits its message by using signs that exhibit a high degree of redundancy such that only one message is decoded among a selection of competing ones.[20]

Integrated process organization of neural information

Quantitative information theoretic methods have been applied in cognitive science to analyze the integrated process organization of neural information in the context of the binding problem in cognitive neuroscience.[21] In this context, either an information-theoretical measure, such as

"functional clusters" (G.M. Edelman's and G. Tononi's "Functional Clustering Model" and "Dynamic Core Hypothesis (DCH)"[22]) or "effective information" (G. Tononi's and O. Sporn's "Information Integration Theory (ITT) of Consciousness"[23][24][25] (see also "integrated information theory")), is defined (on the basis of a reentrant process organization, i.e. the synchronization of neurophysiological activity between groups of neuronal populations), or the measure of the minimization of "free energy" on the basis of statistical methods (K.J. Friston's "free energy principle (FEP)", an information-theoretical measure which states that every adaptive change in a self-organized system leads to a minimization of "free energy", and the "Bayesian Brain Hypothesis"[26][27][28][29][30]).

Miscellaneous applications

Information theory also has applications in gambling, black holes, and bioinformatics.

See also

- Algorithmic probability
- Bayesian inference
- Communication theory
- Constructor theory - a generalization of information theory that includes quantum information
- Formal science
- Inductive probability
- Info-metrics
- Minimum message length
- Minimum description length
- List of important publications
- Philosophy of information

Applications

- Active networking
- Cryptanalysis
- Cryptography
- Cybernetics
- Entropy in thermodynamics and information theory
- Gambling
- Intelligence (information gathering)
- Seismic exploration

History

- Hartley, R.V.L.
- History of information theory
- Shannon, C.E.
- Timeline of information theory
- Yockey, H.P.

Theory

- Coding theory
- Detection theory
- Estimation theory
- Fisher information
- Information algebra
- Information asymmetry
- Information field theory
- Information geometry

- Information theory and measure theory
- Kolmogorov complexity
- List of unsolved problems in information theory
- Logic of information
- Network coding
- Philosophy of information
- Quantum information science
- Source coding

Concepts

- Ban (unit)
- Channel capacity
- Communication channel
- Communication source
- Conditional entropy
- Covert channel
- Data compression
- Decoder
- Differential entropy
- Fungible information
- Information fluctuation complexity
- Information entropy
- Joint entropy
- Kullback–Leibler divergence
- Mutual information
- Pointwise mutual information (PMI)
- Receiver (information theory)
- Redundancy
- Rényi entropy
- Self-information
- Unicity distance
- Variety
- Hamming distance

References

1. "Claude Shannon, pioneered digital information theory" (https://www.fiercetelecom.com/special-report/claude-shannon-pioneered-digital-information-theory). *FierceTelecom*. Retrieved 2021-04-30.
2. Shannon, Claude Elwood (1998). *The mathematical theory of communication* (https://www.worldcat.org/oclc/40716662). Warren Weaver. Urbana: University of Illinois Press. ISBN 0-252-72546-8. OCLC 40716662 (https://www.worldcat.org/oclc/40716662).
3. Burnham, K. P. and Anderson D. R. (2002) *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach, Second Edition* (Springer Science, New York) ISBN 978-0-387-95364-9.
4. F. Rieke; D. Warland; R Ruyter van Steveninck; W Bialek (1997). *Spikes: Exploring the Neural Code*. The MIT press. ISBN 978-0262681087.
5. Delgado-Bonal, Alfonso; Martín-Torres, Javier (2016-11-03). "Human vision is determined based on information theory" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5093619). *Scientific Reports*. **6** (1): 36038. Bibcode:2016NatSR...636038D (https://ui.adsabs.harvard.edu/abs/2016NatSR...636038D). doi:10.1038/srep36038 (https://doi.org/10.1038%2Fsrep36038). ISSN 2045-2322 (https://www.worldcat.org/issn/2045-2322). PMC 5093619 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5093619). PMID 27808236 (https://pubmed.ncbi.nlm.nih.gov/27808236).
6. cf; Huelsenbeck, J. P.; Ronquist, F.; Nielsen, R.; Bollback, J. P. (2001). "Bayesian inference of phylogeny and its impact on evolutionary biology". *Science*. **294** (5550): 2310–2314. Bibcode:2001Sci...294.2310H (https://ui.adsabs.harvard.edu/abs/2001Sci...294.2310H). doi:10.1126/science.1065889 (https://doi.org/10.1126%2Fscience.1065889). PMID 11743192 (https://pubmed.ncbi.nlm.nih.gov/11743192). S2CID 2138288 (https://api.semanticscholar.org/CorpusID:2138288).

7. Allikmets, Rando; Wasserman, Wyeth W.; Hutchinson, Amy; Smallwood, Philip; Nathans, Jeremy; Rogan, Peter K. (1998). "Thomas D. Schneider], Michael Dean (1998) Organization of the ABCR gene: analysis of promoter and splice junction sequences" (http://alum.mit.edu/www/toms/). *Gene*. **215** (1): 111–122. doi:10.1016/s0378-1119(98)00269-8 (https://doi.org/10.1016%2Fs0378-1119%2898%2900269-8). PMID 9666097 (https://pubmed.ncbi.nlm.nih.gov/9666097).
8. Jaynes, E. T. (1957). "Information Theory and Statistical Mechanics" (http://bayes.wustl.edu/). *Phys. Rev*. **106** (4): 620. Bibcode:1957PhRv..106..620J (https://ui.adsabs.harvard.edu/abs/1957PhRv..106..620J). doi:10.1103/physrev.106.620 (https://doi.org/10.1103%2Fphysrev.106.620).
9. Talaat, Khaled; Cowen, Benjamin; Anderoglu, Osman (2020-10-05). "Method of information entropy for convergence assessment of molecular dynamics simulations". *Journal of Applied Physics*. **128** (13): 135102. Bibcode:2020JAP...128m5102T (https://ui.adsabs.harvard.edu/abs/2020JAP...128m5102T). doi:10.1063/5.0019078 (https://doi.org/10.1063%2F5.0019078). OSTI 1691442 (https://www.osti.gov/biblio/1691442). S2CID 225010720 (https://api.semanticscholar.org/CorpusID:225010720).
10. Bennett, Charles H.; Li, Ming; Ma, Bin (2003). "Chain Letters and Evolutionary Histories" (https://web.archive.org/web/20071007041539/http://www.sciamdigital.com/index.cfm?fa=Products.ViewIssuePreview&ARTICLEID_CHAR=08B64096-0772-4904-9D48227D5C9FAC75). *Scientific American*. **288** (6): 76–81. Bibcode:2003SciAm.288f..76B (https://ui.adsabs.harvard.edu/abs/2003SciAm.288f..76B). doi:10.1038/scientificamerican0603-76 (https://doi.org/10.1038%2Fscientificamerican0603-76). PMID 12764940 (https://pubmed.ncbi.nlm.nih.gov/12764940). Archived from the original (http://sciamdigital.com/index.cfm?fa=Products.ViewIssuePreview&ARTICLEID_CHAR=08B64096-0772-4904-9D48227D5C9FAC75) on 2007-10-07. Retrieved 2008-03-11.
11. David R. Anderson (November 1, 2003). "Some background on why people in the empirical sciences may want to better understand the information-theoretic methods" (https://web.archive.org/web/20110723045720/http://aicanderson2.home.comcast.net/~aicanderson2/home.pdf) (PDF). Archived from the original (http://aicanderson2.home.comcast.net/~aicanderson2/home.pdf) (PDF) on July 23, 2011. Retrieved 2010-06-23.
12. Fazlollah M. Reza (1994) [1961]. *An Introduction to Information Theory* (https://books.google.com/books?id=RtzpRAiX6OgC&q=intitle:%22An+Introduction+to+Information+Theory%22++%22entropy+of+a+simple+source%22&pg=PA8). Dover Publications, Inc., New York. ISBN 0-486-68210-2.
13. Robert B. Ash (1990) [1965]. *Information Theory* (https://books.google.com/books?id=ngZhvUfF0UIC&q=intitle:information+intitle:theory+inauthor:ash+conditional+uncertainty&pg=PA16). Dover Publications, Inc. ISBN 0-486-66521-6.
14. Jerry D. Gibson (1998). *Digital Compression for Multimedia: Principles and Standards* (https://books.google.com/books?id=aqQ2Ry6spu0C&q=entropy-rate+conditional&pg=PA56). Morgan Kaufmann. ISBN 1-55860-369-7.
15. Massey, James L. (1990). "Causality, Feedback And Directed Information" (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.5688). CiteSeerX 10.1.1.36.5688 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.5688).
16. Permuter, Haim Henry; Weissman, Tsachy; Goldsmith, Andrea J. (February 2009). "Finite State Channels With Time-Invariant Deterministic Feedback". *IEEE Transactions on Information Theory*. **55** (2): 644–662. arXiv:cs/0608070 (https://arxiv.org/abs/cs/0608070). doi:10.1109/TIT.2008.2009849 (https://doi.org/10.1109%2FTIT.2008.2009849). S2CID 13178 (https://api.semanticscholar.org/CorpusID:13178).
17. Haggerty, Patrick E. (1981). "The corporation and innovation". *Strategic Management Journal*. **2** (2): 97–118. doi:10.1002/smj.4250020202 (https://doi.org/10.1002%2Fsmj.4250020202).
18. Nauta, Doede (1972). *The Meaning of Information*. The Hague: Mouton. ISBN 9789027919960.

19. Nöth, Winfried (January 2012). "Charles S. Peirce's theory of information: a theory of the growth of symbols and of knowledge" (https://edisciplinas.usp.br/mod/resource/view.php?id=2311849). *Cybernetics and Human Knowing*. **19** (1–2): 137–161.
20. Nöth, Winfried (1981). "Semiotics of ideology (https://kobra.uni-kassel.de/bitstream/handle/123456789/2014122246977/semi_2004_002.pdf?sequence=1&isAllowed=y)". *Semiotica*, Issue 148.
21. Maurer, H. (2021). Cognitive Science: Integrative Synchronization Mechanisms in Cognitive Neuroarchitectures of the Modern Connectionism. CRC Press, Boca Raton/FL, chap. 10, ISBN 978-1-351-04352-6. https://doi.org/10.1201/9781351043526
22. Edelman, G.M. and G. Tononi (2000). A Universe of Consciousness: How Matter Becomes Imagination. Basic Books, New York.
23. Tononi, G. and O. Sporns (2003). Measuring information integration. BMC Neuroscience 4: 1-20.
24. Tononi, G. (2004a). An information integration theory of consciousness. BMC Neuroscience 5: 1-22.
25. Tononi, G. (2004b). Consciousness and the brain: theoretical aspects. In: G. Adelman and B. Smith [eds.]: Encyclopedia of Neuroscience. 3rd Ed. Elsevier, Amsterdam, Oxford.
26. Friston, K. and K.E. Stephan (2007). Free-energy and the brain. Synthese 159: 417-458.
27. Friston, K. (2010). The free-energy principle: a unified brain theory. Nature Reviews Neuroscience 11: 127-138.
28. Friston, K., M. Breakstear and G. Deco (2012). Perception and self-organized instability. Frontiers in Computational Neuroscience 6: 1-19.
29. Friston, K. (2013). Life as we know it. Journal of the Royal Society Interface 10: 20130475.
30. Kirchhoff, M., T. Parr, E. Palacios, K. Friston and J. Kiverstein. (2018). The Markov blankets of life: autonomy, active inference and the free energy principle. Journal of the Royal Society Interface 15: 20170792.

Further reading

The classic work

- Shannon, C.E. (1948), "A Mathematical Theory of Communication", *Bell System Technical Journal*, 27, pp. 379–423 & 623–656, July & October, 1948. PDF. (http://math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf)
  Notes and other formats. (https://web.archive.org/web/20150409204946/http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html)
- R.V.L. Hartley, "Transmission of Information" (http://www.dotrose.com/etext/90_Miscellaneous/transmission_of_information_1928b.pdf), *Bell System Technical Journal*, July 1928
- Andrey Kolmogorov (1968), "Three approaches to the quantitative definition of information (https://www.tandfonline.com/doi/pdf/10.1080/00207166808803030)" in International Journal of Computer Mathematics.

Other journal articles

- J. L. Kelly, Jr., Princeton (http://www.princeton.edu/~wbialek/rome/refs/kelly_56.pdf), "A New Interpretation of Information Rate" *Bell System Technical Journal*, Vol. 35, July 1956, pp. 917–26.
- R. Landauer, IEEE.org (http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=615478), "Information is Physical" *Proc. Workshop on Physics and Computation PhysComp'92* (IEEE Comp. Sci.Press, Los Alamitos, 1993) pp. 1–4.

- Landauer, R. (1961). "Irreversibility and Heat Generation in the Computing Process" (http://www.research.ibm.com/journal/rd/441/landauerii.pdf) (PDF). *IBM J. Res. Dev.* **5** (3): 183–191. doi:10.1147/rd.53.0183 (https://doi.org/10.1147%2Frd.53.0183).
- Timme, Nicholas; Alford, Wesley; Flecker, Benjamin; Beggs, John M. (2012). "Multivariate information measures: an experimentalist's perspective". arXiv:1111.6857 (https://arxiv.org/abs/1111.6857) [cs.IT (https://arxiv.org/archive/cs.IT)].

Textbooks on information theory

- Arndt, C. *Information Measures, Information and its Description in Science and Engineering* (Springer Series: Signals and Communication Technology), 2004, ISBN 978-3-540-40855-0
- Ash, RB. *Information Theory*. New York: Interscience, 1965. ISBN 0-470-03445-9. New York: Dover 1990. ISBN 0-486-66521-6
- Gallager, R. *Information Theory and Reliable Communication.* New York: John Wiley and Sons, 1968. ISBN 0-471-29048-3
- Goldman, S. *Information Theory*. New York: Prentice Hall, 1953. New York: Dover 1968 ISBN 0-486-62209-6, 2005 ISBN 0-486-44271-3
- Cover, Thomas; Thomas, Joy A. (2006). *Elements of information theory* (2nd ed.). New York: Wiley-Interscience. ISBN 0-471-24195-4.
- Csiszar, I, Korner, J. *Information Theory: Coding Theorems for Discrete Memoryless Systems* Akademiai Kiado: 2nd edition, 1997. ISBN 963-05-7440-3
- MacKay, David J. C. *Information Theory, Inference, and Learning Algorithms (http://www.inference.phy.cam.ac.uk/mackay/itila/book.html)* Cambridge: Cambridge University Press, 2003. ISBN 0-521-64298-1
- Mansuripur, M. *Introduction to Information Theory*. New York: Prentice Hall, 1987. ISBN 0-13-484668-0
- McEliece, R. *The Theory of Information and Coding*. Cambridge, 2002. ISBN 978-0521831857
- Pierce, JR. "An introduction to information theory: symbols, signals and noise". Dover (2nd Edition). 1961 (reprinted by Dover 1980).
- Reza, F. *An Introduction to Information Theory*. New York: McGraw-Hill 1961. New York: Dover 1994. ISBN 0-486-68210-2
- Shannon, Claude; Weaver, Warren (1949). *The Mathematical Theory of Communication* (http://monoskop.org/images/b/be/Shannon_Claude_E_Weaver_Warren_The_Mathematical_Theory_of_Communication_1963.pdf) (PDF). Urbana, Illinois: University of Illinois Press. ISBN 0-252-72548-4. LCCN 49-11922 (https://lccn.loc.gov/49-11922).
- Stone, JV. Chapter 1 of book "Information Theory: A Tutorial Introduction" (http://jim-stone.staff.shef.ac.uk/BookInfoTheory/InfoTheoryBookMain.html), University of Sheffield, England, 2014. ISBN 978-0956372857.
- Yeung, RW. *A First Course in Information Theory (http://iest2.ie.cuhk.edu.hk/~whyeung/book/)* Kluwer Academic/Plenum Publishers, 2002. ISBN 0-306-46791-7.
- Yeung, RW. *Information Theory and Network Coding (http://iest2.ie.cuhk.edu.hk/~whyeung/book2/)* Springer 2008, 2002. ISBN 978-0-387-79233-0

Other books

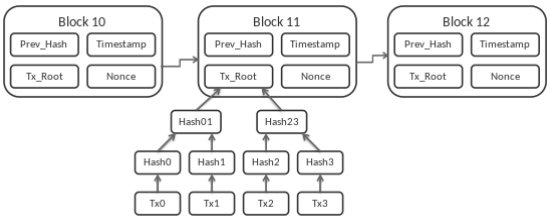
- Leon Brillouin, *Science and Information Theory*, Mineola, N.Y.: Dover, [1956, 1962] 2004. ISBN 0-486-43918-6
- James Gleick, *The Information: A History, a Theory, a Flood*, New York: Pantheon, 2011. ISBN 978-0-375-42372-7

- A. I. Khinchin, *Mathematical Foundations of Information Theory*, New York: Dover, 1957. ISBN 0-486-60434-9
- H. S. Leff and A. F. Rex, Editors, *Maxwell's Demon: Entropy, Information, Computing*, Princeton University Press, Princeton, New Jersey (1990). ISBN 0-691-08727-X
- Robert K. Logan. *What is Information? - Propagating Organization in the Biosphere, the Symbolosphere, the Technosphere and the Econosphere*, Toronto: DEMO Publishing.
- Tom Siegfried, *The Bit and the Pendulum*, Wiley, 2000. ISBN 0-471-32174-5
- Charles Seife, *Decoding the Universe*, Viking, 2006. ISBN 0-670-03441-X
- Jeremy Campbell, *Grammatical Man*, Touchstone/Simon & Schuster, 1982, ISBN 0-671-44062-4
- Henri Theil, *Economics and Information Theory*, Rand McNally & Company - Chicago, 1967.
- Escolano, Suau, Bonev, *Information Theory in Computer Vision and Pattern Recognition (https://www.springer.com/computer/image+processing/book/978-1-84882-296-2)*, Springer, 2009. ISBN 978-1-84882-296-2
- Vlatko Vedral, *Decoding Reality: The Universe as Quantum Information*, Oxford University Press 2010. ISBN 0-19-923769-7

External links

- "Information" (https://www.encyclopediaofmath.org/index.php?title=Information), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Lambert F. L. (1999), "Shuffled Cards, Messy Desks, and Disorderly Dorm Rooms - Examples of Entropy Increase? Nonsense! (http://jchemed.chem.wisc.edu/Journal/Issues/1999/Oct/abs1385.html)", *Journal of Chemical Education*
- IEEE Information Theory Society (http://www.itsoc.org/) and ITSOC Monographs, Surveys, and Reviews (https://www.itsoc.org/resources/surveys)

Blockchain



Bitcoin blockchain structure

A **blockchain** is a type of Digital Ledger Technology (DLT) that consists of growing list of records, called *blocks*, that are securely linked together using cryptography.[1][2][3][4] Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leafs). The timestamp proves that the transaction data existed when the block was created. Since each block contains information about the block previous to it, they effectively form a *chain* (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

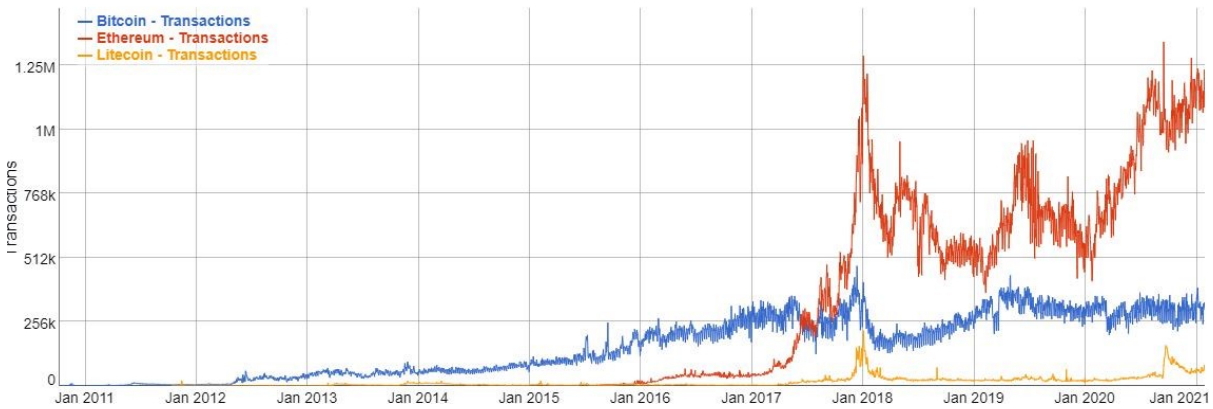
Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.[5]

A blockchain was created by a person (or group of people) using the name (or pseudonym) Satoshi Nakamoto in 2008[6] to serve as the public distributed ledger for bitcoin cryptocurrency transactions, based on previous work by Stuart Haber, W. Scott Stornetta, and Dave Bayer.[7] The identity of Satoshi Nakamoto remains unknown to date. The implementation of the blockchain within bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server. The bitcoin design has inspired other applications[3][2] and blockchains that are readable by the public and are widely used by cryptocurrencies. The blockchain may be considered a type of payment rail.[8]

Private blockchains have been proposed for business use. *Computerworld* called the marketing of such privatized blockchains without a proper security model "snake oil";[9] however, others have argued that permissioned blockchains, if carefully designed, may be more decentralized and therefore more secure in practice than permissionless ones.[4][10]

Contents

History



Bitcoin, Ethereum and Litecoin transactions per day (January 2011 – January 2021)

Cryptographer David Chaum first proposed a blockchain-like protocol in his 1982 dissertation "Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups."[11] Further work on a cryptographically secured chain of blocks was described in 1991 by Stuart Haber and W. Scott Stornetta.[4][12] They wanted to implement a system wherein document timestamps could not be tampered with. In 1992, Haber, Stornetta, and Dave Bayer incorporated Merkle trees into the design, which improved its efficiency by allowing several document certificates to be collected into one block.[4][13] Under their company Surety, their document certificate hashes have been published in *The New York Times* every week since 1995.[14]

The first decentralized blockchain was conceptualized by a person (or group of people) known as Satoshi Nakamoto in 2008. Nakamoto improved the design in an important way using a Hashcash-like method to timestamp blocks without requiring them to be signed by a trusted party and introducing a difficulty parameter to stabilize the rate at which blocks are added to the chain.[4] The design was implemented the following year by Nakamoto as a core component of the cryptocurrency bitcoin, where it serves as the public ledger for all transactions on the network.[3]

In August 2014, the bitcoin blockchain file size, containing records of all transactions that have occurred on the network, reached 20 GB (gigabytes).[15] In January 2015, the size had grown to almost 30 GB, and from January 2016 to January 2017, the bitcoin blockchain grew from 50 GB to 100 GB in size. The ledger size had exceeded 200 GB by early 2020.[16]

The words *block* and *chain* were used separately in Satoshi Nakamoto's original paper, but were eventually popularized as a single word, *blockchain,* by 2016.[17]

According to Accenture, an application of the diffusion of innovations theory suggests that blockchains attained a 13.5% adoption rate within financial services in 2016, therefore reaching the early adopters' phase.[18] Industry trade groups joined to create the Global Blockchain Forum in 2016, an initiative of the Chamber of Digital Commerce.

In May 2018, Gartner found that only 1% of CIOs indicated any kind of blockchain adoption within their organisations, and only 8% of CIOs were in the short-term "planning or [looking at] active experimentation with blockchain".[19] For the year 2019 Gartner reported 5% of CIOs believed blockchain technology was a 'game-changer' for their business.[20]

Structure & Design

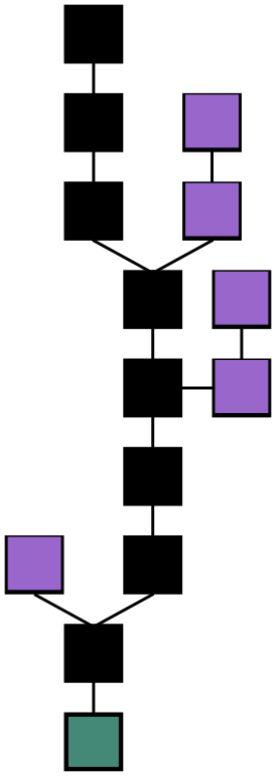
A blockchain is a decentralized, distributed, and often public, digital ledger consisting of records called *blocks* that are used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks.[3][21] This allows the participants to verify and audit transactions independently and relatively inexpensively.[22] A blockchain database is managed autonomously using a peer-to-peer network and a distributed timestamping server. They are authenticated by mass collaboration powered by collective self-interests.[23] Such a design facilitates robust workflow where participants' uncertainty regarding data security is marginal. The use of a blockchain removes the characteristic of infinite reproducibility from a digital asset. It confirms that each unit of value was transferred only once, solving the long-standing problem of double-spending. A blockchain has been described as a *value-exchange protocol*.[24] A blockchain can maintain title rights because, when properly set up to detail the exchange agreement, it provides a record that compels offer and acceptance.

Logically, a blockchain can be seen as consisting of several layers:[25]

- infrastructure (hardware)
- networking (node discovery, information propagation[26] and verification)

- consensus (proof of work, proof of stake)
- data (blocks, transactions)
- application (smart contracts/decentralized applications, if applicable)

Blocks



Blockchain formation. The main chain (black) consists of the longest series of blocks from the genesis block (green) to the current block. Orphan blocks (purple) exist outside of the main chain.

Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree.[3] Each block includes the cryptographic hash of the prior block in the blockchain, linking the two. The linked blocks form a chain.[3] This iterative process confirms the integrity of the previous block, all the way back to the initial block, which is known as the *genesis block* (Block 0).[27][28] To assure the integrity of a block and the data contained in it, the block is usually digitally signed.[29]

Sometimes separate blocks can be produced concurrently, creating a temporary fork. In addition to a secure hash-based history, any blockchain has a specified algorithm for scoring different versions of the history so that one with a higher score can be selected over others. Blocks not selected for inclusion in the chain are called orphan blocks.[28] Peers supporting the database have different versions of the history from time to time. They keep only the highest-scoring version of the database known to them. Whenever a peer receives a higher-scoring version (usually the old version with a single new block added) they extend or overwrite their own database and retransmit the improvement to their peers. There is never an absolute guarantee that any particular entry will remain in the best version of history forever. Blockchains are typically built to add the score of new blocks onto old blocks and are given incentives to extend with new blocks rather than overwrite old blocks. Therefore, the probability of an entry becoming superseded decreases exponentially[30] as more blocks are built on top of it, eventually becoming very low.[3][31]:ch. 08[32] For example, bitcoin uses a proof-of-work system, where the chain with the most cumulative proof-of-work is considered the valid one by the network. There are a number of methods that can be used to demonstrate a sufficient level of computation. Within a blockchain the computation is carried out redundantly rather than in the traditional segregated and parallel manner.[33]

Block time

The *block time* is the average time it takes for the network to generate one extra block in the blockchain. Some blockchains create a new block as frequently as every five seconds.[34] By the time of block completion, the included data becomes verifiable. In cryptocurrency, this is practically when the transaction takes place, so a shorter block time means faster transactions. The block time for Ethereum is set to between 14 and 15 seconds, while for bitcoin it is on average 10 minutes.[35]

Hard forks

A *hard fork* is a rule change such that the software validating according to the old rules will see the blocks produced according to the new rules as invalid. In case of a hard fork, all nodes meant to work in accordance with the new rules need to upgrade their software. If one group of nodes continues to use the old software while the other nodes use the new software, a permanent split can occur.

For example, Ethereum was hard-forked in 2016 to "make whole" the investors in The DAO, which had been hacked by exploiting a vulnerability in its code. In this case, the fork resulted in a split creating Ethereum and Ethereum Classic chains. In 2014 the Nxt community was asked to consider a hard fork that would have led to a rollback of the blockchain records to mitigate the effects of a theft of 50 million NXT from a major cryptocurrency exchange. The hard fork proposal was rejected, and some of the funds were recovered after negotiations and ransom payment. Alternatively, to prevent a permanent split, a majority of nodes using the new software may return to the old rules, as was the case of bitcoin split on 12 March 2013.[36]

A more recent hard-fork example is of Bitcoin in 2017, which resulted in a split creating Bitcoin Cash.[37] The network split was mainly due to a disagreement in how to increase the transactions per second to accommodate for demand.[38]

Decentralization

By storing data across its peer-to-peer network, the blockchain eliminates a number of risks that come with data being held centrally.[3] The decentralized blockchain may use ad hoc message passing and distributed networking. One risk of a lack of decentralization is a so-called "51% attack" where a central entity can gain control of more than half of a network and can manipulate that specific blockchain record at will, allowing double-spending.[39]

Peer-to-peer blockchain networks lack centralized points of vulnerability that computer crackers can exploit; likewise, they have no central point of failure. Blockchain security methods include the use of public-key cryptography.[40]:5 A *public key* (a long, random-looking string of numbers) is an address on the blockchain. Value tokens sent across the network are recorded as belonging to that address. A *private key* is like a password that gives its owner access to their digital assets or the means to otherwise interact with the various capabilities that blockchains now support. Data stored on the blockchain is generally considered incorruptible.[3]

Every node in a decentralized system has a copy of the blockchain. Data quality is maintained by massive database replication[41] and computational trust. No centralized "official" copy exists and no user is "trusted" more than any other.[40] Transactions are broadcast to the network using the software. Messages are delivered on a best-effort basis. Early blockchains rely on energy-intensive mining nodes to validate transactions,[28] add them to the block they are building, and then broadcast the completed block to other nodes.[31]:ch. 08 Blockchains use various time-stamping schemes, such as proof-of-work, to serialize changes.[42] Later consensus methods include proof of stake.[28] The growth of a decentralized blockchain is accompanied by the risk of centralization because the computer resources required to process larger amounts of data become more expensive.[43]

Openness

Open blockchains are more user-friendly than some traditional ownership records, which, while open to the public, still require physical access to view. Because all early blockchains were permissionless, controversy has arisen over the blockchain definition. An issue in this ongoing debate is whether a private system with verifiers tasked and authorized (permissioned) by a central authority should be considered a blockchain.[44][45][46][47][48] Proponents of permissioned or private chains argue that the term "blockchain" may be applied to any **data structure** that batches data into time-stamped blocks. These blockchains serve as a distributed version of multiversion concurrency control (MVCC) in databases.[49] Just as MVCC prevents two transactions from concurrently modifying a single object in a database, blockchains prevent two transactions from spending the same single output in a blockchain.[50]:30–31 Opponents say that

permissioned systems resemble traditional corporate databases, not supporting decentralized data verification, and that such systems are not hardened against operator tampering and revision.[44][46] Nikolai Hampton of *Computerworld* said that "many in-house blockchain solutions will be nothing more than cumbersome databases," and "without a clear security model, proprietary blockchains should be eyed with suspicion."[9][51]

Permissionless (public) blockchain

An advantage to an open, permissionless, or public, blockchain network is that guarding against bad actors is not required and no access control is needed.[30] This means that applications can be added to the network without the approval or trust of others, using the blockchain as a transport layer.[30]

Bitcoin and other cryptocurrencies currently secure their blockchain by requiring new entries to include proof of work. To prolong the blockchain, bitcoin uses Hashcash puzzles. While Hashcash was designed in 1997 by Adam Back, the original idea was first proposed by Cynthia Dwork and Moni Naor and Eli Ponyatovski in their 1992 paper "Pricing via Processing or Combatting Junk Mail".

In 2016, venture capital investment for blockchain-related projects was weakening in the USA but increasing in China.[52] Bitcoin and many other cryptocurrencies use open (public) blockchains. As of April 2018, bitcoin has the highest market capitalization.

Permissioned (private) blockchain

Permissioned blockchains use an access control layer to govern who has access to the network.[53] In contrast to public blockchain networks, validators on private blockchain networks are vetted by the network owner. They do not rely on anonymous nodes to validate transactions nor do they benefit from the network effect.[54] Permissioned blockchains can also go by the name of 'consortium' blockchains.[55] It has been argued that permissioned blockchains can guarantee a certain level of decentralization, if carefully designed, as opposed to permissionless blockchains, which are often centralized in practice.[10]

Disadvantages of permissioned blockchain

Nikolai Hampton pointed out in *Computerworld* that "There is also no need for a '51 percent' attack on a private blockchain, as the private blockchain (most likely) already controls 100 percent of all block creation resources. If you could attack or damage the blockchain creation tools on a private corporate server, you could effectively control 100 percent of their network and alter transactions however you wished."[9] This has a set of particularly profound adverse implications during a financial crisis or debt crisis like the financial crisis of 2007–08, where politically powerful actors may make decisions that favor some groups at the expense of others,[56] and "the bitcoin blockchain is protected by the massive group mining effort. It's unlikely that any private blockchain will try to protect records using gigawatts of computing power — it's time-consuming and expensive."[9] He also said, "Within a private blockchain there is also no 'race'; there's no incentive to use more power or discover blocks faster than competitors. This means that many in-house blockchain solutions will be nothing more than cumbersome databases."[9]

Blockchain analysis

The analysis of public blockchains has become increasingly important with the popularity of bitcoin, Ethereum, litecoin and other cryptocurrencies.[57] A blockchain, if it is public, provides anyone who wants access to observe and analyse the chain data, given one has the know-how. The process of understanding and accessing the flow of crypto has been an issue for many cryptocurrencies, crypto exchanges and banks.[58][59] The reason for this is accusations of blockchain-enabled cryptocurrencies enabling illicit dark market trade of drugs, weapons, money laundering, etc.[60] A common belief has been that cryptocurrency is private and untraceable, thus leading many actors to use it for illegal purposes. This is changing and now specialised tech companies provide blockchain tracking services, making crypto exchanges, law-enforcement and banks more aware of what is happening with crypto funds and fiat-crypto exchanges. The development, some argue, has led criminals to prioritise the use of new cryptos such as Monero.[61][62][63] The question is about the public accessibility of blockchain data and the personal privacy of the very same data. It is a key debate in cryptocurrency and ultimately in the blockchain.[64]

Standardisation

In April 2016, Standards Australia submitted a proposal to the International Organization for Standardization to consider developing standards to support blockchain technology. This proposal resulted in the creation of ISO Technical Committee 307, Blockchain and Distributed Ledger Technologies.[65] The technical committee has working groups relating to blockchain terminology, reference architecture, security and privacy, identity, smart contracts, governance and interoperability for blockchain and DLT, as well as standards specific to industry sectors and generic government requirements.[66] More than 50 countries are participating in the standardization process together with external liaisons such as the Society for Worldwide Interbank Financial Telecommunication (SWIFT), the European Commission, the International Federation of Surveyors, the International Telecommunication Union (ITU) and the United Nations Economic Commission for Europe (UNECE).[66]

Many other national standards bodies and open standards bodies are also working on blockchain standards.[67] These include the National Institute of Standards and Technology[68] (NIST), the European Committee for Electrotechnical Standardization[69] (CENELEC), the Institute of Electrical and Electronics Engineers[70] (IEEE), the Organization for the Advancement of Structured Information Standards (OASIS), and some individual participants in the Internet Engineering Task Force[71] (IETF).

Centralized Blockchain

Although most of blockchain implementation are decentralized and distributed, Oracle launched a centralized blockchain table feature in Oracle 21c database. The Blockchain Table in Oracle 21c database is a centralized blockchain which provide immutable feature. Compared to decentralized blockchains, centralized blockchains normally can provide a higher throughput and lower latency of transactions than consensus-based distributed blockchains.[72][73]

Types

Currently, there are at least four types of blockchain networks — public blockchains, private blockchains, consortium blockchains and hybrid blockchains.

Public blockchains

A public blockchain has absolutely no access restrictions. Anyone with an Internet connection can send transactions to it as well as become a validator (i.e., participate in the execution of a consensus protocol).[74] Usually, such networks offer economic incentives for those who secure them and utilize some type of a Proof of Stake or Proof of Work algorithm.

Some of the largest, most known public blockchains are the bitcoin blockchain and the Ethereum blockchain.

Private blockchains

A private blockchain is permissioned.[53] One cannot join it unless invited by the network administrators. Participant and validator access is restricted. To distinguish between open blockchains and other peer-to-peer decentralized database applications that are not open ad-hoc compute clusters, the terminology Distributed Ledger (DLT) is normally used for private blockchains.

Hybrid blockchains

A hybrid blockchain has a combination of centralized and decentralized features.[75] The exact workings of the chain can vary based on which portions of centralization and decentralization are used.

Sidechains

A sidechain is a designation for a blockchain ledger that runs in parallel to a primary blockchain.[76][77] Entries from the primary blockchain (where said entries typically represent digital assets) can be linked to and from the sidechain; this allows the sidechain to otherwise operate independently of the primary blockchain (e.g., by using an alternate means of record keeping, alternate consensus algorithm, etc.).[78]

Uses

Blockchain technology can be integrated into multiple areas. The primary use of blockchains is as a distributed ledger for cryptocurrencies such as bitcoin; there were also a few other operational products that had matured from proof of concept by late 2016.[52] As of 2016, some businesses have been testing the technology and conducting low-level implementation to gauge blockchain's effects on organizational efficiency in their back office.[79]

Bitcoin's transactions are recorded on a publicly viewable blockchain.

In 2019, it was estimated that around $2.9 billion were invested in blockchain technology, which represents an 89% increase from the year prior. Additionally, the International Data Corp has estimated that corporate investment into blockchain technology will reach $12.4 billion by 2022.[80] Furthermore, According to PricewaterhouseCoopers (PwC), the second-largest professional services network in the world, blockchain technology has the potential to generate an annual business value of more than $3 trillion by 2030. PwC's estimate is further augmented by a 2018 study that they have conducted, in

which PwC surveyed 600 business executives and determined that 84% have at least some exposure to utilizing blockchain technology, which indicts a significant demand and interest in blockchain technology.[81]

Individual use of blockchain technology has also greatly increased since 2016. According to statistics in 2020, there were more than 40 million blockchain wallets in 2020 in comparison to around 10 million blockchain wallets in 2016.[82]

Blockchain technology in sustainable management[83]

Cryptocurrencies

Most cryptocurrencies use blockchain technology to record transactions. For example, the bitcoin network and Ethereum network are both based on blockchain. On 8 May 2018 Facebook confirmed that it would open a new blockchain group[84] which would be headed by David Marcus, who previously was in charge of Messenger. Facebook's planned cryptocurrency platform, Libra (now known as Diem), was formally announced on June 18, 2019.[85][86]

The criminal enterprise Silk Road, which operated on Tor, utilized cryptocurrency for payments, some of which the US federal government has seized through research on the blockchain and forfeiture.[87]

Governments have mixed policies on the legality of their citizens or banks owning cryptocurrencies. China implements blockchain technology in several industries including a national digital currency which launched in 2020.[88] To strengthen their respective currencies, Western governments including the European Union and the United States have initiated similar projects.[89]

Smart contracts

Blockchain-based smart contracts are proposed contracts that can be partially or fully executed or enforced without human interaction.[90] One of the main objectives of a smart contract is automated escrow. A key feature of smart contracts is that they do not need a trusted third party (such as a trustee) to act as an intermediary between contracting entities — the blockchain network executes the contract on its own. This may reduce friction between entities when transferring value and could subsequently open the door to a higher level of transaction automation.[91] An IMF staff discussion from 2018 reported that smart contracts based on blockchain technology might reduce moral hazards and optimize the use of contracts in general. But "no viable smart contract systems have yet emerged." Due to the lack of widespread use their legal status was unclear.[92][93]

Financial services

According to *Reason*, many banks have expressed interest in implementing distributed ledgers for use in banking and are cooperating with companies creating private blockchains,[94][95][96] and according to a September 2016 IBM study, this is occurring faster than expected.[97]

Banks are interested in this technology not least because it has the potential to speed up back office settlement systems.[98] Moreover, as the blockchain industry has reached early maturity institutional appreciation has grown that it is, practically speaking, the infrastructure of a whole new financial industry, with all the implications which that entails.[99]

Banks such as UBS are opening new research labs dedicated to blockchain technology in order to explore how blockchain can be used in financial services to increase efficiency and reduce costs.[100][101]

Berenberg, a German bank, believes that blockchain is an "overhyped technology" that has had a large number of "proofs of concept", but still has major challenges, and very few success stories.[102]

The blockchain has also given rise to initial coin offerings (ICOs) as well as a new category of digital asset called security token offerings (STOs), also sometimes referred to as digital security offerings (DSOs).[103] STO/DSOs may be conducted privately or on public, regulated stock exchange and are used to tokenize traditional assets such as company shares as well as more innovative ones like intellectual property, real estate,[104] art, or individual products. A number of companies are active in this space providing services for compliant tokenization, private STOs, and public STOs.

Games

Blockchain technology, such as cryptocurrencies and non-fungible tokens (NFTs), has been used in video games for monetization. Many live-service games offer in-game customization options, such as character skins or other in-game items, which the players can earn and trade with other players using in-game currency. Some games also allow for trading of virtual items using real-world currency, but this may be illegal in some countries where video games are seen as akin to gambling, and has led to gray market issues such as skin gambling, and thus publishers typically have shied away from allowing players to earn real-world funds from games.[105] Blockchain games typically allow players to trade these in-game items for cryptocurrency, which can then be exchanged for money.[106]

The first known game to use blockchain technologies was *CryptoKitties*, launched in November 2017, where the player would purchase NFTs with Ethereum cryptocurrency, each NFT consisting of a virtual pet that the player could breed with others to create offspring with combined traits as new NFTs.[107][106] The game made headlines in December 2017 when one virtual pet sold for more than US$100,000.[108] *CryptoKitties* also illustrated scalability problems for games on Ethereum when it created significant congestion on the Ethereum network in early 2018 with approximately 30% of all Ethereum transactions being for the game.[109][110]

By the early 2020s, there had not been a breakout success in video games using blockchain, as these games tend to focus on using blockchain for speculation instead of more traditional forms of gameplay, which offers limited appeal to most players. Such games also represent a high risk to investors as their revenues can be difficult to predict.[106] However, limited successes of some games, such as *Axie Infinity* during the COVID-19 pandemic, and corporate plans towards metaverse content, refueled interest in the area of GameFi, a term describing the intersection of video games and financing typically backed by blockchain currency, in the second half of 2021.[111] Several major publishers, including Ubisoft, Electronic Arts, and Take Two Interactive, have stated that blockchain and NFT-based games are under serious consideration for their companies in the future.[112]

In October 2021, Valve Corporation banned blockchain games, including those using cryptocurrency and NFTs, from being hosted on its Steam digital storefront service, which is widely used for personal computer gaming, claiming that this was an extension of their policy banning games that offered in-game items with real-world value. Valve's prior history with gambling, specifically skin gambling, was speculated to be a factor in the decision to ban blockchain games.[113] Journalists and players responded positively to Valve's decision as blockchain and NFT

games have a reputation for scams and fraud among most PC gamers,[105][113] Epic Games, which runs the Epic Games Store in competition to Steam, said that they would be open to accepted blockchain games, in the wake of Valve's refusal.[114]

Supply chain

There have been several different efforts to employ blockchains in supply chain management.

- Shipping industry — incumbent shipping companies and startups have begun to leverage blockchain technology to facilitate the emergence of a blockchain-based platform ecosystem that would create value across the global shipping supply chains.[115]
- **Precious commodities mining** — Blockchain technology has been used for tracking the origins of gemstones and other precious commodities. In 2016, *The Wall Street Journal* reported that the blockchain technology company Everledger was partnering with IBM's blockchain-based tracking service to trace the origin of diamonds to ensure that they were ethically mined.[116] As of 2019, the Diamond Trading Company (DTC) has been involved in building a diamond trading supply chain product called Tracr.[117]
- **Food supply** — As of 2018, Walmart and IBM were running a trial to use a blockchain-backed system for supply chain monitoring for lettuce and spinach — all nodes of the blockchain were administered by Walmart and were located on the IBM cloud.[118]
- Fashion industry — There is an opaque relationship between brands, distributors, and customers in the fashion industry, which will prevent the sustainable and stable development of the fashion industry. Blockchain makes up for this shortcoming and makes information transparent, solving the difficulty of sustainable development of the industry.[119]

Domain names

There are several different efforts to offer domain name services via the blockchain. These domain names can be controlled by the use of a private key, which purports to allow for uncensorable websites. This would also bypass a registrar's ability to suppress domains used for fraud, abuse, or illegal content.[120]

Namecoin is a cryptocurrency that supports the ".bit" top-level domain (TLD). Namecoin was forked from bitcoin in 2011. The .bit TLD is not sanctioned by ICANN, instead requiring an alternative DNS root.[120] As of 2015, it was used by 28 websites, out of 120,000 registered names.[121] Namecoin was dropped by OpenNIC in 2019, due to malware and potential other legal issues.[122] Other blockchain alternatives to ICANN include The Handshake Network,[121] EmerDNS, and Unstoppable Domains.[120]

Specific TLDs include ".eth", ".luxe", and ".kred", which are associated with the Ethereum blockchain through the Ethereum Name Service (ENS). The .kred TLD also acts as an alternative to conventional cryptocurrency wallet addresses, as a convenience for transferring cryptocurrency.[123]

Other uses

Blockchain technology can be used to create a permanent, public, transparent ledger system for compiling data on sales, tracking digital use and payments to content creators, such as wireless users[124] or musicians.[125] The Gartner 2019 CIO Survey reported 2% of higher education respondents had launched blockchain projects and another 18% were planning academic projects in the next 24 months.[126] In 2017, IBM partnered with ASCAP and PRS for Music to adopt

blockchain technology in music distribution.[127] Imogen Heap's Mycelia service has also been proposed as a blockchain-based alternative "that gives artists more control over how their songs and associated data circulate among fans and other musicians."[128][129]

New distribution methods are available for the insurance industry such as peer-to-peer insurance, parametric insurance and microinsurance following the adoption of blockchain.[130][131] The sharing economy and IoT are also set to benefit from blockchains because they involve many collaborating peers.[132] The use of blockchain in libraries is being studied with a grant from the U.S. Institute of Museum and Library Services.[133]

Other blockchain designs include Hyperledger, a collaborative effort from the Linux Foundation to support blockchain-based distributed ledgers, with projects under this initiative including Hyperledger Burrow (by Monax) and Hyperledger Fabric (spearheaded by IBM).[134][135][136] Another is Quorum, a permissionable private blockchain by JPMorgan Chase with private storage, used for contract applications.[137]

Oracle introduced a blockchain table feature in its Oracle 21c database.[72][73]

Blockchain is also being used in peer-to-peer energy trading.[138][139][140]

Blockchain could be used in detecting counterfeits by associating unique identifiers to products, documents and shipments, and storing records associated with transactions that cannot be forged or altered.[141][142] It is however argued that blockchain technology needs to be supplemented with technologies that provide a strong binding between physical objects and blockchain systems.[143] The EUIPO established an Anti-Counterfeiting Blockathon Forum, with the objective of "defining, piloting and implementing" an anti-counterfeiting infrastructure at the European level.[144][145] The Dutch Standardisation organisation NEN uses blockchain together with QR Codes to authenticate certificates.[146]

2022 Jan 30 Beijing and Shanghai are among the cities designated by China to trial blockchain applications.[147]

Blockchain interoperability

With the increasing number of blockchain systems appearing, even only those that support cryptocurrencies, blockchain interoperability is becoming a topic of major importance. The objective is to support transferring assets from one blockchain system to another blockchain system. Wegner[148] stated that "interoperability is the ability of two or more software components to cooperate despite differences in language, interface, and execution platform". The objective of blockchain interoperability is therefore to support such cooperation among blockchain systems, despite those kinds of differences.

There are already several blockchain interoperability solutions available.[149] They can be classified into three categories: cryptocurrency interoperability approaches, blockchain engines, and blockchain connectors.

Several individual IETF participants produced the draft of a blockchain interoperability architecture.[150]

Energy consumption concerns

Some cryptocurrencies use blockchain mining — the peer-to-peer computer computations by which transactions are validated and verified. This requires a large amount of energy. In June 2018 the Bank for International Settlements criticized the use of public proof-of-work blockchains for their high energy consumption.[151][152][153]

Early concern over the high energy consumption was a factor in later blockchains such as Cardano (2017), Solana (2020) and Polkadot (2020) adopting the less energy-intensive proof-of-stake model. Researchers have estimated that Bitcoin consumes 100,000 times as much energy as proof-of-stake networks.[154][155]

In 2021, a study by Cambridge University determined that Bitcoin (at 121 terawatt-hours per year) used more electricity than Argentina (at 121TWh) and the Netherlands (109TWh).[156] According to Digiconomist, one bitcoin transaction required 708 kilowatt-hours of electrical energy, the amount an average U.S. household consumed in 24 days.[157]

In February 2021, U.S. Treasury secretary Janet Yellen called Bitcoin "an extremely inefficient way to conduct transactions", saying "the amount of energy consumed in processing those transactions is staggering".[158] In March 2021, Bill Gates stated that "Bitcoin uses more electricity per transaction than any other method known to mankind", adding "It's not a great climate thing."[159]

Nicholas Weaver, of the International Computer Science Institute at the University of California, Berkeley, examined blockchain's online security, and the energy efficiency of proof-of-work public blockchains, and in both cases found it grossly inadequate.[160][161] The 31TWh-45TWh of electricity used for bitcoin in 2018 produced 17-23 million tonnes of CO_2.[162][163] By 2022, the University of Cambridge and Digiconomist estimated that the two largest proof-of-work blockchains, Bitcoin and Ethereum, together used twice as much electricity in one year as the whole of Sweden, leading to the release of up to 120 million tonnes of CO_2 each year.[164]

Some cryptocurrency developers are considering moving from the proof-of-work model to the proof-of-stake model.[165]

Academic research

In October 2014, the MIT Bitcoin Club, with funding from MIT alumni, provided undergraduate students at the Massachusetts Institute of Technology access to $100 of bitcoin. The adoption rates, as studied by Catalini and Tucker (2016), revealed that when people who typically adopt technologies early are given delayed access, they tend to reject the technology.[166] Many universities have founded departments focusing on crypto and blockchain, including MIT, in 2017. In the same year, Edinburgh became "one of the first big European universities to launch a blockchain course", according to the *Financial Times*.[167]

Blockchain panel discussion at the first IEEE Computer Society TechIgnite conference

Adoption decision

Motivations for adopting blockchain technology (an aspect of innovation adoptation) have been investigated by researchers. For example, Janssen, et al. provided a framework for analysis,[168] and Koens & Poll pointed out that adoption could be heavily driven by non-technical factors.[169] Based on behavioral models, Li[170] has discussed the differences between adoption at the individual level and organizational levels.

Collaboration

Scholars in business and management have started studying the role of blockchains to support collaboration.[171][172] It has been argued that blockchains can foster both cooperation (i.e., prevention of opportunistic behavior) and coordination (i.e., communication and information sharing). Thanks to reliability, transparency, traceability of records, and information immutability, blockchains facilitate collaboration in a way that differs both from the traditional use of contracts and from relational norms. Contrary to contracts, blockchains do not directly rely on the legal system to enforce agreements.[173] In addition, contrary to the use of relational norms, blockchains do not require a trust or direct connections between collaborators.

Blockchain and internal audit

External video

Blockchain Basics & Cryptography (https://www.youtube.com/watch?v=0UvVOMZqpEA), Gary Gensler, Massachusetts Institute of Technology, 0:30[174]

The need for internal audits to provide effective oversight of organizational efficiency will require a change in the way that information is accessed in new formats.[175] Blockchain adoption requires a framework to identify the risk of exposure associated with transactions using blockchain. The Institute of Internal Auditors has identified the need for internal auditors to address this transformational technology. New methods are required to develop audit plans that identify threats and risks. The Internal Audit Foundation study, *Blockchain and Internal Audit,* assesses these factors.[176] The American Institute of Certified Public Accountants has outlined new roles for auditors as a result of blockchain.[177]

Journals

In September 2015, the first peer-reviewed academic journal dedicated to cryptocurrency and blockchain technology research, *Ledger*, was announced. The inaugural issue was published in December 2016.[178] The journal covers aspects of mathematics, computer science, engineering, law, economics and philosophy that relate to cryptocurrencies such as bitcoin.[179][180]

The journal encourages authors to digitally sign a file hash of submitted papers, which are then timestamped into the bitcoin blockchain. Authors are also asked to include a personal bitcoin address on the first page of their papers for non-repudiation purposes.[181]

See also

- Changelog – a record of all notable changes made to a project
- Checklist – an informational aid used to reduce failure
- Economics of digitization
- Privacy and blockchain
- Version control – a record of all changes (mostly of software project) in a form of a graph

References

1. Morris, David Z. (15 May 2016). "Leaderless, Blockchain-Based Venture Capital Fund Raises $100 Million, And Counting" (http://fortune.com/2016/05/15/leaderless-blockchain-vc-fund/). *Fortune*. Archived (https://web.archive.org/web/20160521015602/http://fortune.com/2016/05/15/leaderless-blockchain-vc-fund/) from the original on 21 May 2016. Retrieved 23 May 2016.

2. Popper, Nathan (21 May 2016). "A Venture Fund With Plenty of Virtual Capital, but No Capitalist" (https://www.nytimes.com/2016/05/22/business/dealbook/crypto-ether-bitcoin-currency.html). *The New York Times*. Archived (https://web.archive.org/web/20160522034932/http://www.nytimes.com/2016/05/22/business/dealbook/crypto-ether-bitcoin-currency.html) from the original on 22 May 2016. Retrieved 23 May 2016.
3. "Blockchains: The great chain of being sure about things" (https://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable). *The Economist*. 31 October 2015. Archived (https://web.archive.org/web/20160703000844/http://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable) from the original on 3 July 2016. Retrieved 18 June 2016. "The technology behind bitcoin lets people who do not know or trust each other build a dependable ledger. This has implications far beyond the crypto currency."
4. Narayanan, Arvind; Bonneau, Joseph; Felten, Edward; Miller, Andrew; Goldfeder, Steven (2016). *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton: Princeton University Press. ISBN 978-0-691-17169-2.
5. Iansiti, Marco; Lakhani, Karim R. (January 2017). "The Truth About Blockchain" (https://hbr.org/2017/01/the-truth-about-blockchain). *Harvard Business Review*. Harvard University. Archived (https://web.archive.org/web/20170118052537/https://hbr.org/2017/01/the-truth-about-blockchain) from the original on 18 January 2017. Retrieved 17 January 2017. "The technology at the heart of bitcoin and other virtual currencies, blockchain is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way."
6. Satoshi Nakamoto (October 2008). "Bitcoin: A Peer-to-Peer Electronic Cash System" (https://cryptocurrencyworks.com/.res/doc/Bitcoin-SNakamoto-Oct-2008.pdf) (PDF). Retrieved 29 July 2022.
7. "The World's Oldest Blockchain Has Been Hiding in the New York Times Since 1995" (https://www.vice.com/en/article/j5nzx4/what-was-the-first-blockchain). *www.vice.com*. Retrieved 9 October 2021.
8. "Blockchain may finally disrupt payments from Micropayments to credit cards to SWIFT" (https://dailyfintech.com/2018/02/10/bitcoin-will-finally-disrupt-the-credit-card-rails/). *dailyfintech.com*. 10 February 2018. Archived (https://web.archive.org/web/20180927005613/https://dailyfintech.com/2018/02/10/bitcoin-will-finally-disrupt-the-credit-card-rails/) from the original on 27 September 2018. Retrieved 18 November 2018.
9. Hampton, Nikolai (5 September 2016). "Understanding the blockchain hype: Why much of it is nothing more than snake oil and spin" (https://www2.computerworld.com.au/article/606253/understanding-blockchain-hype-why-much-it-nothing-more-than-snake-oil-spin/). *Computerworld*. Archived (https://web.archive.org/web/20160906171838/http://www.computerworld.com.au/article/606253/understanding-blockchain-hype-why-much-it-nothing-more-than-snake-oil-spin/) from the original on 6 September 2016. Retrieved 5 September 2016.
10. Bakos, Yannis; Halaburda, Hanna; Mueller-Bloch, Christoph (February 2021). "When Permissioned Blockchains Deliver More Decentralization Than Permissionless". *Communications of the ACM*. **64** (2): 20–22. doi:10.1145/3442371 (https://doi.org/10.1145%2F3442371). S2CID 231704491 (https://api.semanticscholar.org/CorpusID:231704491).
11. Sherman, Alan T.; Javani, Farid; Zhang, Haibin; Golaszewski, Enis (January 2019). "On the Origins and Variations of Blockchain Technologies". *IEEE Security Privacy*. **17** (1): 72–77. arXiv:1810.06130 (https://arxiv.org/abs/1810.06130). doi:10.1109/MSEC.2019.2893730 (https://doi.org/10.1109%2FMSEC.2019.2893730). ISSN 1558-4046 (https://www.worldcat.org/issn/1558-4046). S2CID 53114747 (https://api.semanticscholar.org/CorpusID:53114747).
12. Haber, Stuart; Stornetta, W. Scott (January 1991). "How to time-stamp a digital document". *Journal of Cryptology*. **3** (2): 99–111. CiteSeerX 10.1.1.46.8740 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.8740). doi:10.1007/bf00196791 (https://doi.org/10.1007%2Fbf00196791). S2CID 14363020 (https://api.semanticscholar.org/CorpusID:14363020).

13. Bayer, Dave; Haber, Stuart; Stornetta, W. Scott (March 1992). *Improving the Efficiency and Reliability of Digital Time-Stamping*. *Sequences*. Vol. 2. pp. 329–334. CiteSeerX 10.1.1.71.4891 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.4891). doi:10.1007/978-1-4613-9323-8_24 (https://doi.org/10.1007%2F978-1-4613-9323-8_24). ISBN 978-1-4613-9325-2.
14. "The World's Oldest Blockchain Has Been Hiding in the New York Times Since 1995" (https://www.vice.com/en/article/j5nzx4/what-was-the-first-blockchain). *www.vice.com*. Retrieved 9 October 2021.
15. Nian, Lam Pak; Chuen, David LEE Kuo (2015). "A Light Touch of Regulation for Virtual Currencies". In Chuen, David LEE Kuo (ed.). *Handbook of Digital Currency: Bitcoin, Innovation, Financial Instruments, and Big Data*. Academic Press. p. 319. ISBN 978-0-12-802351-8.
16. "Blockchain Size" (https://www.blockchain.com/en/charts/blocks-size?scale=1&timespan=all&showDataPoints=true). Archived (https://web.archive.org/web/20200519024720/https://www.blockchain.com/en/charts/blocks-size?scale=1&timespan=all&showDataPoints=true) from the original on 19 May 2020. Retrieved 25 February 2020.
17. Johnsen, Maria (12 May 2020). *Blockchain in Digital Marketing: A New Paradigm of Trust* (https://books.google.com/books?id=uVbjDwAAQBAJ&pg=PA6). Maria Johnsen. p. 6. ISBN 979-8-6448-7308-1.
18. "The future of blockchain in 8 charts" (http://raconteur.net/business/the-future-of-blockchain-in-8-charts). *Raconteur*. 27 June 2016. Archived (https://web.archive.org/web/20161202234125/http://raconteur.net/business/the-future-of-blockchain-in-8-charts) from the original on 2 December 2016. Retrieved 3 December 2016.
19. "Hype Killer - Only 1% of Companies Are Using Blockchain, Gartner Reports | Artificial Lawyer" (https://www.artificiallawyer.com/2018/05/04/hype-killer-only-1-of-companies-are-using-blockchain-gartner-reports/). *Artificial Lawyer*. 4 May 2018. Archived (https://web.archive.org/web/20180522111623/https://www.artificiallawyer.com/2018/05/04/hype-killer-only-1-of-companies-are-using-blockchain-gartner-reports/) from the original on 22 May 2018. Retrieved 22 May 2018.
20. Kasey Panetta. (31 October 2018). "Digital Business: CIO Agenda 2019: Exploit Transformational Technologies." Gartner website (https://www.gartner.com/smarterwithgartner/cio-agenda-2019-exploit-transformational-technologies/) Retrieved 27 March 2021.
21. Armstrong, Stephen (7 November 2016). "Move over Bitcoin, the blockchain is only just getting started" (https://www.wired.co.uk/article/unlock-the-blockchain). *Wired*. Archived (https://web.archive.org/web/20161108130946/http://www.wired.co.uk/article/unlock-the-blockchain) from the original on 8 November 2016. Retrieved 9 November 2016.
22. Catalini, Christian; Gans, Joshua S. (23 November 2016). "Some Simple Economics of the Blockchain" (http://www.nber.org/papers/w22952.pdf) (PDF). *SSRN*. doi:10.2139/ssrn.2874598 (https://doi.org/10.2139%2Fssrn.2874598). hdl:1721.1/130500 (https://hdl.handle.net/1721.1%2F130500). S2CID 46904163 (https://api.semanticscholar.org/CorpusID:46904163). SSRN 2874598 (https://ssrn.com/abstract=2874598). Archived (https://web.archive.org/web/20200306014931/https://www.nber.org/papers/w22952.pdf) (PDF) from the original on 6 March 2020. Retrieved 16 September 2019.
23. Tapscott, Don; Tapscott, Alex (8 May 2016). "Here's Why Blockchains Will Change the World" (http://fortune.com/2016/05/08/why-blockchains-will-change-the-world/). *Fortune*. Archived (https://web.archive.org/web/20161113134748/http://fortune.com/2016/05/08/why-blockchains-will-change-the-world/) from the original on 13 November 2016. Retrieved 16 November 2016.
24. Bheemaiah, Kariappa (January 2015). "Block Chain 2.0: The Renaissance of Money" (https://www.wired.com/insights/2015/01/block-chain-2-0/). *Wired*. Archived (https://web.archive.org/web/20161114001509/https://www.wired.com/insights/2015/01/block-chain-2-0/) from the original on 14 November 2016. Retrieved 13 November 2016.

25. Chen, Huashan; Pendleton, Marcus; Njilla, Laurent; Xu, Shouhuai (12 June 2020). "A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses" (https://arxiv.org/abs/1908.04507). *ACM Computing Surveys*. **53** (3): 3–4. arXiv:1908.04507 (https://arxiv.org/abs/1908.04507). doi:10.1145/3391195 (https://doi.org/10.1145%2F3391195). ISSN 0360-0300 (https://www.worldcat.org/issn/0360-0300). S2CID 199551841 (https://api.semanticscholar.org/CorpusID:199551841).
26. Shishir, Bhatia (2 February 2006). *Structured Information Flow (SIF) Framework for Automating End-to-End Information Flow for Large Organizations* (https://vtechworks.lib.vt.edu/handle/10919/31148) (Thesis). Virginia Tech.
27. "Genesis Block Definition" (https://www.investopedia.com/terms/g/genesis-block.asp). *Investopedia*. Retrieved 10 August 2022.
28. Bhaskar, Nirupama Devi; Chuen, David LEE Kuo (2015). "Bitcoin Mining Technology". *Handbook of Digital Currency*. pp. 45–65. doi:10.1016/B978-0-12-802117-0.00003-5 (https://doi.org/10.1016%2FB978-0-12-802117-0.00003-5). ISBN 978-0-12-802117-0.
29. Knirsch, Unterweger & Engel 2019, p. 2.
30. Antonopoulos, Andreas (20 February 2014). "Bitcoin security model: trust by computation" (http://radar.oreilly.com/2014/02/bitcoin-security-model-trust-by-computation.html). *Radar*. O'Reilly. Archived (https://web.archive.org/web/20161031132328/http://radar.oreilly.com/2014/02/bitcoin-security-model-trust-by-computation.html) from the original on 31 October 2016. Retrieved 19 November 2016.
31. Antonopoulos, Andreas M. (2014). *Mastering Bitcoin. Unlocking Digital Cryptocurrencies* (https://web.archive.org/web/20161201131627/http://chimera.labs.oreilly.com/books/1234000001802/ch01.html). Sebastopol, CA: O'Reilly Media. ISBN 978-1449374037. Archived from the original (http://chimera.labs.oreilly.com/books/1234000001802/ch01.html) on 1 December 2016. Retrieved 3 November 2015.
32. Nakamoto, Satoshi (October 2008). "Bitcoin: A Peer-to-Peer Electronic Cash System" (https://bitcoin.org/bitcoin.pdf) (PDF). bitcoin.org. Archived (https://web.archive.org/web/20140320135003/https://bitcoin.org/bitcoin.pdf) (PDF) from the original on 20 March 2014. Retrieved 28 April 2014.
33. "Permissioned Blockchains" (https://monax.io/explainers/permissioned_blockchains/). *Explainer*. Monax. Archived (https://web.archive.org/web/20161120154948/https://monax.io/explainers/permissioned_blockchains/) from the original on 20 November 2016. Retrieved 20 November 2016.
34. Strydom, Moses; Buckley, Sheryl (July 2019). *AI and Big Data's Potential for Disruptive Innovation* (http://www.igi-global.com/book/big-data-potential-disruptive-innovation/222833). IGI Global. ISBN 978-1-5225-9687-5.
35. Kumar, Randhir; Tripathi, Rakesh (November 2019). "Implementation of Distributed File Storage and Access Framework using IPFS and Blockchain". *2019 Fifth International Conference on Image Information Processing (ICIIP)*. IEEE: 246–251. doi:10.1109/iciip47207.2019.8985677 (https://doi.org/10.1109%2Ficiip47207.2019.8985677). ISBN 978-1-7281-0899-5. S2CID 211119043 (https://api.semanticscholar.org/CorpusID:211119043).
36. Lee, Timothy (12 March 2013). "Major glitch in Bitcoin network sparks sell-off; price temporarily falls 23%" (https://arstechnica.com/business/2013/03/major-glitch-in-bitcoin-network-sparks-sell-off-price-temporarily-falls-23/). Arstechnica. Archived (https://web.archive.org/web/20130420050926/http://arstechnica.com/business/2013/03/major-glitch-in-bitcoin-network-sparks-sell-off-price-temporarily-falls-23/) from the original on 20 April 2013. Retrieved 25 February 2018.
37. Smith, Oli (21 January 2018). "Bitcoin price RIVAL: Cryptocurrency 'faster than bitcoin' will CHALLENGE market leaders" (https://www.express.co.uk/finance/city/907536/cryptocurrency-bitcoin-market-leader-cash-ripple-ethereum). *Express*. Retrieved 6 April 2021.
38. "Bitcoin split in two, here's what that means" (https://money.cnn.com/2017/08/01/technology/business/bitcoin-cash-new-currency/index.html). *CNN*. 1 August 2017. Retrieved 7 April 2021.

39. "Bitcoin Spinoff Hacked in Rare '51% Attack' " (https://fortune.com/2018/05/29/bitcoin-gold-hack/). *Fortune*. Retrieved 28 April 2021.
40. Brito, Jerry; Castillo, Andrea (2013). Bitcoin: A Primer for Policymakers (http://mercatus.org/sites/default/files/Brito_BitcoinPrimer.pdf) (PDF) (Report). Fairfax, VA: Mercatus Center, George Mason University. Archived (https://web.archive.org/web/20130921060724/http://mercatus.org/sites/default/files/Brito_BitcoinPrimer.pdf) (PDF) from the original on 21 September 2013. Retrieved 22 October 2013.
41. Raval, Siraj (2016). *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology* (https://books.google.com/books?id=fvywDAAAQBAJ). O'Reilly Media, Inc. pp. 1 (https://books.google.com/books?id=fvywDAAAQBAJ&pg=PA1)–2 (https://books.google.com/books?id=fvywDAAAQBAJ). ISBN 978-1-4919-2452-5.
42. Kopfstein, Janus (12 December 2013). "The Mission to Decentralize the Internet" (https://www.newyorker.com/tech/elements/the-mission-to-decentralize-the-internet). *The New Yorker*. Archived (https://web.archive.org/web/20141231031044/http://www.newyorker.com/tech/elements/the-mission-to-decentralize-the-internet) from the original on 31 December 2014. Retrieved 30 December 2014. "The network's 'nodes' — users running the bitcoin software on their computers — collectively check the integrity of other nodes to ensure that no one spends the same coins twice. All transactions are published on a shared public ledger, called the 'block chain.'"
43. Gervais, Arthur; Karame, Ghassan O.; Capkun, Vedran; Capkun, Srdjan. "Is Bitcoin a Decentralized Currency?" (https://www.infoq.com/articles/is-bitcoin-a-decentralized-currency/). *InfoQ*. InfoQ & IEEE computer society. Archived (https://web.archive.org/web/20161010042659/https://www.infoq.com/articles/is-bitcoin-a-decentralized-currency/) from the original on 10 October 2016. Retrieved 11 October 2016.
44. Voorhees, Erik (30 October 2015). "It's All About the Blockchain" (https://web.archive.org/web/20151101235750/http://moneyandstate.com/its-all-about-the-blockchain/). *Money and State*. Archived from the original (http://moneyandstate.com/its-all-about-the-blockchain/) on 1 November 2015. Retrieved 2 November 2015.
45. Reutzel, Bailey (13 July 2015). "A Very Public Conflict Over Private Blockchains" (http://www.paymentssource.com/news/technology/a-very-public-confluct-over-private-blockchains-3021831-1.html). *PaymentsSource*. New York, NY: SourceMedia, Inc. Archived (https://web.archive.org/web/20160421112045/http://www.paymentssource.com/news/technology/a-very-public-confluct-over-private-blockchains-3021831-1.html) from the original on 21 April 2016. Retrieved 18 June 2016.
46. Casey MJ (15 April 2015). "Moneybeat/BitBeat: Blockchains Without Coins Stir Tensions in Bitcoin Community" (https://blogs.wsj.com/moneybeat/2015/04/14/bitbeat-blockchains-without-coins-stir-tensions-in-bitcoin-community/). *The Wall Street Journal*. Archived (https://web.archive.org/web/20160610224428/http://blogs.wsj.com/moneybeat/2015/04/14/bitbeat-blockchains-without-coins-stir-tensions-in-bitcoin-community/) from the original on 10 June 2016. Retrieved 18 June 2016.
47. "The 'Blockchain Technology' Bandwagon Has A Lesson Left To Learn" (http://news.dinbits.com/2015/11/the-blockchain-technology-bandwagon-has.html). *dinbits.com*. 3 November 2015. Archived (https://web.archive.org/web/20160629234654/http://news.dinbits.com/2015/11/the-blockchain-technology-bandwagon-has.html) from the original on 29 June 2016. Retrieved 18 June 2016.
48. DeRose, Chris (26 June 2015). "Why the Bitcoin Blockchain Beats Out Competitors" (http://www.americanbanker.com/bankthink/why-the-bitcoin-blockchain-beats-out-competitors-1075100-1.html). *American Banker*. Archived (https://web.archive.org/web/20160330060709/http://www.americanbanker.com/bankthink/why-the-bitcoin-blockchain-beats-out-competitors-1075100-1.html) from the original on 30 March 2016. Retrieved 18 June 2016.

49. Greenspan, Gideon (19 July 2015). "Ending the bitcoin vs blockchain debate" (http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/). *multichain.com*. Archived (https://web.archive.org/web/20160608070620/http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/) from the original on 8 June 2016. Retrieved 18 June 2016.
50. Tapscott, Don; Tapscott, Alex (May 2016). *The Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. ISBN 978-0-670-06997-2.
51. Barry, Levine (11 June 2018). "A new report bursts the blockchain bubble" (https://martechtoday.com/a-new-report-bursts-the-blockchain-bubble-216959). MarTech. Archived (https://web.archive.org/web/20180713232406/https://martechtoday.com/a-new-report-bursts-the-blockchain-bubble-216959) from the original on 13 July 2018. Retrieved 13 July 2018.
52. Ovenden, James. "Blockchain Top Trends In 2017" (https://channels.theinnovationenterprise.com/articles/blockchain-top-trends-in-2017). The Innovation Enterprise. Archived (https://web.archive.org/web/20161130143727/https://channels.theinnovationenterprise.com/articles/blockchain-top-trends-in-2017) from the original on 30 November 2016. Retrieved 4 December 2016.
53. Bob Marvin (30 August 2017). "Blockchain: The Invisible Technology That's Changing the World" (http://au.pcmag.com/amazon-web-services/46389/feature/blockchain-the-invisible-technology-thats-changing-the-world). *PC MAG Australia*. ZiffDavis, LLC. Archived (https://web.archive.org/web/20170925180800/http://au.pcmag.com/amazon-web-services/46389/feature/blockchain-the-invisible-technology-thats-changing-the-world) from the original on 25 September 2017. Retrieved 25 September 2017.
54. Prisco, Giulio. "Sandia National Laboratories Joins the War on Bitcoin Anonymity" (https://bitcoinmagazine.com/culture/sandia-national-laboratories-joins-the-war-on-bitcoin-anonymity-1472151009). *Bitcoin Magazine: Bitcoin News, Articles, Charts, and Guides*. Retrieved 20 May 2022.
55. "Blockchains & Distributed Ledger Technologies" (https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/). *BlockchainHub*. Retrieved 20 May 2022.
56. O'Keeffe, M.; Terzi, A. (7 July 2015). "The political economy of financial crisis policy" (http://bruegel.org/2015/07/the-political-economy-of-financial-crisis-policy/). *Bruegel*. Archived (https://web.archive.org/web/20180519025919/http://bruegel.org/2015/07/the-political-economy-of-financial-crisis-policy/) from the original on 19 May 2018. Retrieved 8 May 2018.
57. Dr Garrick Hileman & Michel Rauchs (2017). "GLOBAL CRYPTOCURRENCY BENCHMARKING STUDY" (https://cdn.crowdfundinsider.com/wp-content/uploads/2017/04/Global-Cryptocurrency-Benchmarking-Study.pdf) (PDF). *Cambridge Centre for Alternative Finance*. University of Cambridge Judge Business School. Archived (https://web.archive.org/web/20190515100308/https://cdn.crowdfundinsider.com/wp-content/uploads/2017/04/Global-Cryptocurrency-Benchmarking-Study.pdf) (PDF) from the original on 15 May 2019. Retrieved 15 May 2019 – via crowdfundinsider.
58. Raymaekers, Wim (March 2015). "Cryptocurrency Bitcoin: Disruption, challenges and opportunities" (https://www.ingentaconnect.com/content/hsp/jpss/2015/00000009/00000001/art00005). *Journal of Payments Strategy & Systems*. **9** (1): 30–46. Archived (https://web.archive.org/web/20190515100259/https://www.ingentaconnect.com/content/hsp/jpss/2015/00000009/00000001/art00005) from the original on 15 May 2019. Retrieved 15 May 2019.
59. "Why Crypto Companies Still Can't Open Checking Accounts" (https://www.bloomberg.com/news/articles/2019-03-03/why-crypto-companies-still-can-t-open-checking-accounts). 3 March 2019. Archived (https://web.archive.org/web/20190604091359/https://www.bloomberg.com/news/articles/2019-03-03/why-crypto-companies-still-can-t-open-checking-accounts) from the original on 4 June 2019. Retrieved 4 June 2019.
60. Christian Brenig, Rafael Accorsi & Günter Müller (Spring 2015). "Economic Analysis of Cryptocurrency Backed Money Laundering" (https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1019&context=ecis2015_cr). *Association for Information Systems AIS Electronic Library (AISeL)*. Archived (https://web.archive.org/web/20190828223207/https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1019&context=ecis2015_cr) from the original on 28 August 2019. Retrieved 15 May 2019.

61. Greenberg, Andy (25 January 2017). "Monero, the Drug Dealer's Cryptocurrency of Choice, Is on Fire" (https://www.wired.com/2017/01/monero-drug-dealers-cryptocurrency-choice-fire/). *Wired*. ISSN 1059-1028 (https://www.worldcat.org/issn/1059-1028). Archived (https://web.archive.org/web/20181210020727/https://www.wired.com/2017/01/monero-drug-dealers-cryptocurrency-choice-fire/) from the original on 10 December 2018. Retrieved 15 May 2019.
62. Orcutt, Mike. "It's getting harder to hide money in Bitcoin" (https://www.technologyreview.com/s/608763/criminals-thought-bitcoin-was-the-perfect-hiding-place-they-thought-wrong/). *MIT Technology Review*. Retrieved 15 May 2019.
63. "Explainer: 'Privacy coin' Monero offers near total anonymity" (https://uk.reuters.com/article/us-crypto-currencies-altcoins-explainer-idUKKCN1SL0F0). *Reuters*. 15 May 2019. Archived (https://web.archive.org/web/20190515075334/https://uk.reuters.com/article/us-crypto-currencies-altcoins-explainer-idUKKCN1SL0F0) from the original on 15 May 2019. Retrieved 15 May 2019.
64. "An Untraceable Currency? Bitcoin Privacy Concerns - FinTech Weekly" (https://magazine.fintechweekly.com/articles/an-untraceable-currency-bitcoin-privacy-concerns). *FinTech Magazine Article*. 7 April 2018. Archived (https://web.archive.org/web/20190515100301/https://magazine.fintechweekly.com/articles/an-untraceable-currency-bitcoin-privacy-concerns) from the original on 15 May 2019. Retrieved 15 May 2019.
65. "Blockchain" (https://www.standards.org.au/engagement-events/flagship-projects/blockchain). *standards.org.au*. Standards Australia. Retrieved 21 June 2021.
66. "ISO/TC 307 Blockchain and distributed ledger technologies" (https://www.iso.org/committee/6266604.html). *iso.org*. ISO. Retrieved 21 June 2021.
67. Deshmukh, Sumedha; Boulais, Océane; Koens, Tommy. "Global Standards Mapping Initiative: An overview of blockchain technical standards" (http://www3.weforum.org/docs/WEF_GSMI_Technical_Standards_2020.pdf) (PDF). *weforum.org*. World Economic Forum. Retrieved 23 June 2021.
68. "Blockchain Overview" (https://www.nist.gov/blockchain). *NIST*. 25 September 2019. Retrieved 21 June 2021.
69. "CEN and CENELEC publish a White Paper on standards in Blockchain & Distributed Ledger Technologies" (https://www.cencenelec.eu/news/brief_news/Pages/TN-2018-085.aspx). *cencenelec.eu*. CENELEC. Retrieved 21 June 2021.
70. "Standards" (https://blockchain.ieee.org/standards). *ieee.org*. IEEE Blockchain. Retrieved 21 June 2021.
71. Hardjono, Thomas. "An Interoperability Architecture for Blockchain/DLT Gateways" (https://www.ietf.org/id/draft-hardjono-blockchain-interop-arch-02.txt). *ietf.org*. IETF. Retrieved 21 June 2021.
72. "Details: Oracle Blockchain Table" (https://docs.oracle.com/en/database/oracle/oracle-database/21/nfcon/details-oracle-blockchain-table-282449857.html). Archived (https://web.archive.org/web/20210120232657/https://docs.oracle.com/en/database/oracle/oracle-database/21/nfcon/details-oracle-blockchain-table-282449857.html) from the original on 20 January 2021. Retrieved 1 January 2022.
73. "Oracle Blockchain Table" (https://docs.oracle.com/en/database/oracle/oracle-database/21/nfcon/oracle-blockchain-table-268779556.html). Archived (https://web.archive.org/web/20210516130824/https://docs.oracle.com/en/database/oracle/oracle-database/21/nfcon/oracle-blockchain-table-268779556.html) from the original on 16 May 2021. Retrieved 1 January 2022.
74. "How Companies Can Leverage Private Blockchains to Improve Efficiency and Streamline Business Processes" (https://perfectial.com/blog/leveraging-private-blockchains-improve-efficiency-streamline-business-processes/). *Perfectial*.
75. [Distributed Ledger Technology: Hybrid Approach, Front-to-Back Designing and Changing Trade Processing Infrastructure, By Martin Walker, First published:, 24 OCT 2018 ISBN 978-1-78272-389-9]

76. Siraj Raval (18 July 2016). *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology* (https://books.google.com/books?id=fvywDAAAQBAJ&pg=PA22). "O'Reilly Media, Inc.". pp. 22–. ISBN 978-1-4919-2452-5.
77. Niaz Chowdhury (16 August 2019). *Inside Blockchain, Bitcoin, and Cryptocurrencies* (https://books.google.com/books?id=_HatDwAAQBAJ&pg=PA22). CRC Press. pp. 22–. ISBN 978-1-00-050770-6.
78. U.S. Patent 10,438,290 (https://patents.google.com/patent/US10438290)
79. Katie Martin (27 September 2016). "CLS dips into blockchain to net new currencies" (https://www.ft.com/content/c905b6fc-4dd2-3170-9d2a-c79cdbb24f16). *Financial Times*. Archived (https://web.archive.org/web/20161109152317/https://www.ft.com/content/c905b6fc-4dd2-3170-9d2a-c79cdbb24f16) from the original on 9 November 2016. Retrieved 7 November 2016.
80. Castillo, Michael (16 April 2019). "blockchain 50: Billion Dollar Babies" (https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain-50-billion-dollar-babies/). *Financial Website*. SourceMedia. Retrieved 1 February 2021.
81. Davies, Steve (2018). "PwC's Global Blockchain Survey" (https://www.pwc.com/gx/en/industries/technology/blockchain/blockchain-in-business.html). *Financial Website*. SourceMedia. Retrieved 1 February 2021.
82. Liu, Shanhong (13 March 2020). "Blockchain - Statistics & Facts" (https://www.statista.com/topics/5122/blockchain/#dossierSummary__chapter6). *Statistics Website*. SourceMedia. Retrieved 17 February 2021.
83. Du, Wenbo; Ma, Xiaozhi; Yuan, Hongping; Zhu, Yue (1 August 2022). "Blockchain technology-based sustainable management research: the status quo and a general framework for future application" (https://doi.org/10.1007/s11356-022-21761-2). *Environmental Science and Pollution Research*. **29** (39): 58648–58663. doi:10.1007/s11356-022-21761-2 (https://doi.org/10.1007%2Fs11356-022-21761-2). ISSN 1614-7499 (https://www.worldcat.org/issn/1614-7499). PMC 9261142 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9261142). PMID 35794327 (https://pubmed.ncbi.nlm.nih.gov/35794327).
84. Wagner, Kurt (8 May 2018). "Facebook is making its biggest executive shuffle in company history" (https://www.recode.net/2018/5/8/17330226/facebook-reorg-mark-zuckerberg-whatsapp-messenger-ceo-blockchain). *Recode*. Archived (https://web.archive.org/web/20180722053519/https://www.recode.net/2018/5/8/17330226/facebook-reorg-mark-zuckerberg-whatsapp-messenger-ceo-blockchain) from the original on 22 July 2018. Retrieved 25 September 2018.
85. Isaac, Mike; Popper, Nathaniel (18 June 2019). "Facebook Plans Global Financial System Based on Cryptocurrency" (https://www.nytimes.com/2019/06/18/technology/facebook-cryptocurrency-libra.html). *The New York Times*. Archived (https://web.archive.org/web/20200519040623/https://www.nytimes.com/2019/06/18/technology/facebook-cryptocurrency-libra.html) from the original on 19 May 2020. Retrieved 18 June 2019.
86. Constine, Josh (18 June 2019). "Facebook announces Libra cryptocurrency: All you need to know" (https://techcrunch.com/2019/06/18/facebook-libra/). *TechCrunch*. Archived (https://web.archive.org/web/20190619003937/https://techcrunch.com/2019/06/18/facebook-libra/) from the original on 19 June 2019. Retrieved 19 June 2019.
87. KPIX-TV. (5 November 2020). "Silk Road: Feds Seize $1 Billion In Bitcoins Linked To Infamous Silk Road Dark Web Case; 'Where Did The Money Go'". KPIX website (https://sanfrancisco.cbslocal.com/2020/11/05/silk-road-feds-seize-1-billion-in-bitcoins-linked-to-infamous-silk-road-dark-web-case-where-did-the-money-go/) Retrieved 28 March 2021.
88. Aditi Kumar and Eric Rosenbach. (20 May 2020). "Could China's Digital Currency Unseat the Dollar?: American Economic and Geopolitical Power Is at Stake". Foreign Affairs website (https://www.foreignaffairs.com/articles/china/2020-05-20/could-chinas-digital-currency-unseat-dollar) Retrieved 31 March 2021.
89. Staff. (16 February 2021). "The Economist Explains: What is the fuss over central-bank digital currencies?" The Economist website (https://www.economist.com/the-economist-explains/2021/02/16/what-is-the-fuss-over-central-bank-digital-currencies) Retrieved 1 April 2021.

90. Franco, Pedro (2014). *Understanding Bitcoin: Cryptography, Engineering and Economics* (https://books.google.com/books?id=YHfCBwAAQBAJ). John Wiley & Sons. p. 9. ISBN 978-1-119-01916-9. Archived (https://web.archive.org/web/20170214204859/https://books.google.com/books?id=YHfCBwAAQBAJ) from the original on 14 February 2017. Retrieved 4 January 2017 – via Google Books.
91. Casey M (16 July 2018). *The impact of blockchain technology on finance : a catalyst for change* (https://www.worldcat.org/oclc/1059331326). London, UK. ISBN 978-1-912179-15-2. OCLC 1059331326 (https://www.worldcat.org/oclc/1059331326).
92. Governatori, Guido; Idelberger, Florian; Milosevic, Zoran; Riveret, Regis; Sartor, Giovanni; Xu, Xiwei (2018). "On legal contracts, imperative and declarative smart contracts, and blockchain systems". *Artificial Intelligence and Law*. **26** (4): 33. doi:10.1007/s10506-018-9223-3 (https://doi.org/10.1007%2Fs10506-018-9223-3). S2CID 3663005 (https://api.semanticscholar.org/CorpusID:3663005).
93. *Virtual Currencies and Beyond: Initial Considerations* (https://www.imf.org/external/pubs/ft/sdn/2016/sdn1603.pdf) (PDF). IMF Discussion Note. International Monetary Fund. 2016. p. 23. ISBN 978-1-5135-5297-2. Archived (https://web.archive.org/web/20180414210721/https://www.imf.org/external/pubs/ft/sdn/2016/sdn1603.pdf) (PDF) from the original on 14 April 2018. Retrieved 19 April 2018.
94. Epstein, Jim (6 May 2016). "Is Blockchain Technology a Trojan Horse Behind Wall Street's Walled Garden?" (http://reason.com/reasontv/2016/05/06/bitcoin-consensus-blockchain-wall-street). *Reason*. Archived (https://web.archive.org/web/20160708170429/http://reason.com/reasontv/2016/05/06/bitcoin-consensus-blockchain-wall-street) from the original on 8 July 2016. Retrieved 29 June 2016. "mainstream misgivings about working with a system that's open for anyone to use. Many banks are partnering with companies building so-called private blockchains that mimic some aspects of Bitcoin's architecture except they're designed to be closed off and accessible only to chosen parties. ... [but some believe] that open and permission-less blockchains will ultimately prevail even in the banking sector simply because they're more efficient."
95. Redrup, Yolanda (29 June 2016). "ANZ backs private blockchain, but won't go public" (http://www.afr.com/technology/anz-backs-private-blockchain-but-wont-go-public-20160629-gpuf9z). *Australia Financial Review*. Archived (https://web.archive.org/web/20160703103852/http://www.afr.com/technology/anz-backs-private-blockchain-but-wont-go-public-20160629-gpuf9z) from the original on 3 July 2016. Retrieved 7 July 2016. "Blockchain networks can be either public or private. Public blockchains have many users and there are no controls over who can read, upload or delete the data and there are an unknown number of pseudonymous participants. In comparison, private blockchains also have multiple data sets, but there are controls in place over who can edit data and there are a known number of participants."
96. Shah, Rakesh (1 March 2018). "How Can The Banking Sector Leverage Blockchain Technology?" (http://www.postboxcommunications.com/blog/can-banking-sector-leverage-blockchain-technology/). *PostBox Communications*. PostBox Communications Blog. Archived (https://web.archive.org/web/20180317232141/http://www.postboxcommunications.com/blog/can-banking-sector-leverage-blockchain-technology/) from the original on 17 March 2018. "Banks preferably have a notable interest in utilizing Blockchain Technology because it is a great source to avoid fraudulent transactions. Blockchain is considered hassle free, because of the extra level of security it offers."
97. Kelly, Jemima (28 September 2016). "Banks adopting blockchain 'dramatically faster' than expected: IBM" (https://www.reuters.com/article/us-tech-blockchain-ibm-idUSKCN11Y28D). *Reuters*. Archived (https://web.archive.org/web/20160928163048/http://www.reuters.com/article/us-tech-blockchain-ibm-idUSKCN11Y28D) from the original on 28 September 2016. Retrieved 28 September 2016.
98. Arnold, Martin (23 September 2013). "IBM in blockchain project with China UnionPay" (https://www.ft.com/content/719f4e7e-80e1-11e6-bc52-0c7211ef3198). *Financial Times*. Archived (https://web.archive.org/web/20161109152822/https://www.ft.com/content/719f4e7e-80e1-11e6-bc52-0c7211ef3198) from the original on 9 November 2016. Retrieved 7 November 2016.

99. Ravichandran, Arvind; Fargo, Christopher; Kappos, David; Portilla, David; Buretta, John; Ngo, Minh Van; Rosenthal-Larrea, Sasha (28 January 2022). "Blockchain in the Banking Sector: A Review of the Landscape and Opportunities" (https://corpgov.law.harvard.edu/2022/01/28/blockchain-in-the-banking-sector-a-review-of-the-landscape-and-opportunities/).
100. "UBS leads team of banks working on blockchain settlement system" (https://www.reuters.com/article/us-banks-blockchain-ubs-idUSKCN10Z147). *Reuters*. 24 August 2016. Archived (https://web.archive.org/web/20170519073429/http://www.reuters.com/article/us-banks-blockchain-ubs-idUSKCN10Z147) from the original on 19 May 2017. Retrieved 13 May 2017.
101. "Cryptocurrency Blockchain" (https://www.capgemini.com/beyond-the-buzz/cryptocurrency-blockchain). *capgemini.com*. Archived (https://web.archive.org/web/20161205202317/https://www.capgemini.com/beyond-the-buzz/cryptocurrency-blockchain) from the original on 5 December 2016. Retrieved 13 May 2017.
102. Kelly, Jemima (31 October 2017). "Top banks and R3 build blockchain-based payments system" (https://www.reuters.com/article/us-banks-blockchain-r3/top-banks-and-r3-build-blockchain-based-payments-system-idUSKBN1D00ZB). *Reuters*. Archived (https://web.archive.org/web/20180710011735/https://www.reuters.com/article/us-banks-blockchain-r3/top-banks-and-r3-build-blockchain-based-payments-system-idUSKBN1D00ZB) from the original on 10 July 2018. Retrieved 9 July 2018.
103. "Are Token Assests the Securities of Tomorrow?" (https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/technology/lu-token-assets-securities-tomorrow.pdf) (PDF). *Deloitte*. February 2019. Archived (https://web.archive.org/web/20190623225917/https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/technology/lu-token-assets-securities-tomorrow.pdf) (PDF) from the original on 23 June 2019. Retrieved 26 September 2019.
104. Hammerberg, Jeff (7 November 2021). "Potential impact of blockchain on real estate" (https://www.washingtonblade.com/2021/11/07/potential-impact-of-blockchain-on-real-estate/). *Washington Blade*.
105. Clark, Mitchell (15 October 2021). "Valve bans blockchain games and NFTs on Steam, Epic will try to make it work" (https://www.theverge.com/2021/10/15/22728425/valve-steam-blockchain-nft-crypto-ban-games-age-of-rust). *The Verge*. Retrieved 8 November 2021.
106. Mozuch, Mo (29 April 2021). "Blockchain Games Twist The Fundamentals Of Online Gaming" (https://www.inverse.com/gaming/blockchain-games-online-gaming). *Inverse*. Retrieved 4 November 2021.
107. "Internet firms try their luck at blockchain games" (https://asiatimes.com/article/internet-firms-try-luck-blockchain-games/). *Asia Times*. 22 February 2018. Retrieved 28 February 2018.
108. Evelyn Cheng (6 December 2017). "Meet CryptoKitties, the $100,000 digital beanie babies epitomizing the cryptocurrency mania" (https://www.cnbc.com/2017/12/06/meet-cryptokitties-the-new-digital-beanie-babies-selling-for-100k.html). CNBC. Archived (https://web.archive.org/web/20181120115903/https://www.cnbc.com/2017/12/06/meet-cryptokitties-the-new-digital-beanie-babies-selling-for-100k.html) from the original on 20 November 2018. Retrieved 28 February 2018.
109. Laignee Barron (13 February 2018). "CryptoKitties is Going Mobile. Can Ethereum Handle the Traffic?" (http://fortune.com/2018/02/13/cryptokitties-ethereum-ios-launch-china-ether/). *Fortune*. Archived (https://web.archive.org/web/20181028041832/http://fortune.com/2018/02/13/cryptokitties-ethereum-ios-launch-china-ether/) from the original on 28 October 2018. Retrieved 30 September 2018.
110. "CryptoKitties craze slows down transactions on Ethereum" (https://www.bbc.com/news/technology-42237162). 12 May 2017. Archived (https://web.archive.org/web/20180112143517/http://www.bbc.com/news/technology-42237162) from the original on 12 January 2018.
111. Wells, Charlie; Egkolfopoulou, Misrylena (30 October 2021). "Into the Metaverse: Where Crypto, Gaming and Capitalism Collide" (https://www.bloomberg.com/news/features/2021-10-30/what-is-the-metaverse-where-crypto-nft-capitalism-collide-in-games-like-axie). *Bloomberg News*. Retrieved 11 November 2021.

112. Orland, Kyle (4 November 2021). "Big-name publishers see NFTs as a big part of gaming's future" (https://arstechnica.com/gaming/2021/11/big-name-publishers-see-nfts-as-a-big-part-of-gamings-future/). *Ars Technica*. Retrieved 4 November 2021.
113. Knoop, Joseph (15 October 2021). "Steam bans all games with NFTs or cryptocurrency" (https://www.pcgamer.com/steam-bans-nfts-cryptocurrencies-blockchain/). *PC Gamer*. Retrieved 8 November 2021.
114. Clark, Mitchell (15 October 2021). "Epic says it's 'open' to blockchain games after Steam bans them" (https://www.theverge.com/2021/10/15/22729050/epic-game-store-open-to-blockchain-cryptocurrency-nft-games). *The Verge*. Retrieved 11 November 2021.
115. Jovanovic, Marin; Kostić, Nikola; Sebastian, Ina; Sedej, Tomaz (2022). "Managing a blockchain-based platform ecosystem for industry-wide adoption: The case of TradeLens". *Technological Forecasting and Social Change*. doi:10.1016/j.techfore.2022.121981 (https://doi.org/10.1016%2Fj.techfore.2022.121981). ISSN 0040-1625 (https://www.worldcat.org/issn/0040-1625).
116. Nash, Kim S. (14 July 2016). "IBM Pushes Blockchain into the Supply Chain" (https://www.wsj.com/articles/ibm-pushes-blockchain-into-the-supply-chain-1468528824). *The Wall Street Journal*. Archived (https://web.archive.org/web/20160718032702/http://www.wsj.com/articles/ibm-pushes-blockchain-into-the-supply-chain-1468528824) from the original on 18 July 2016. Retrieved 24 July 2016.
117. Gstettner, Stefan (30 July 2019). "How Blockchain Will Redefine Supply Chain Management" (https://knowledge.wharton.upenn.edu/article/blockchain-supply-chain-management). *Knowledge@Wharton*. The Wharton School of the University of Pennsylvania. Retrieved 28 August 2020.
118. Corkery, Michael; Popper, Nathaniel (24 September 2018). "From Farm to Blockchain: Walmart Tracks Its Lettuce" (https://www.nytimes.com/2018/09/24/business/walmart-blockchain-lettuce.html). *The New York Times*. Archived (https://web.archive.org/web/20181205103719/https://www.nytimes.com/2018/09/24/business/walmart-blockchain-lettuce.html) from the original on 5 December 2018. Retrieved 5 December 2018.
119. "Blockchain basics: Utilizing blockchain to improve sustainable supply chains in fashion" (https://www.emerald.com/insight/content/doi/10.1108/SD-03-2021-0028/full/html). *Strategic Direction*. **37** (5): 25–27. 8 June 2021. doi:10.1108/SD-03-2021-0028 (https://doi.org/10.1108%2FSD-03-2021-0028). ISSN 0258-0543 (https://www.worldcat.org/issn/0258-0543). S2CID 241322151 (https://api.semanticscholar.org/CorpusID:241322151).
120. Sanders, James; August 28 (28 August 2019). "Blockchain-based Unstoppable Domains is a rehash of a failed idea" (https://www.techrepublic.com/article/blockchain-based-unstoppable-domains-is-a-rehash-of-a-failed-idea/). *TechRepublic*. Archived (https://web.archive.org/web/20191119015118/https://www.techrepublic.com/article/blockchain-based-unstoppable-domains-is-a-rehash-of-a-failed-idea/) from the original on 19 November 2019. Retrieved 16 April 2020.
121. Orcutt, Mike (4 June 2019). "The ambitious plan to reinvent how websites get their names" (https://www.technologyreview.com/2019/06/04/239039/the-ambitious-plan-to-make-the-internets-phone-book-more-trustworthy/). *MIT Technology Review*. Retrieved 17 May 2021.
122. Cimpanu, Catalin (17 July 2019). "OpenNIC drops support for .bit domain names after rampant malware abuse" (https://www.zdnet.com/article/opennic-drops-support-for-bit-domain-names-after-rampant-malware-abuse/). *ZDNet*. Retrieved 17 May 2021.
123. ".Kred launches as dual DNS and ENS domain" (https://domainnamewire.com/2020/03/06/kred-blockchain-domain/). *Domain Name Wire | Domain Name News*. 6 March 2020. Archived (https://web.archive.org/web/20200308175635/https://domainnamewire.com/2020/03/06/kred-blockchain-domain/) from the original on 8 March 2020. Retrieved 16 April 2020.
124. K. Kotobi, and S. G. Bilen, "Secure Blockchains for Dynamic Spectrum Access : A Decentralized Database in Moving Cognitive Radio Networks Enhances Security and User Access" (http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8269834), IEEE Vehicular Technology Magazine, 2018.

125. "Blockchain Could Be Music's Next Disruptor" (http://fortune.com/2016/09/22/blockchain-music-disruption/). 22 September 2016. Archived (https://web.archive.org/web/20160923180800/http://fortune.com/2016/09/22/blockchain-music-disruption/) from the original on 23 September 2016.
126. Susan Moore. (16 October 2019). "Digital Business: 4 Ways Blockchain Will Transform Higher Education". Gartner website (https://www.gartner.com/smarterwithgartner/4-ways-blockchain-will-transform-higher-education/) Retrieved 27 March 2021.
127. "ASCAP, PRS and SACEM Join Forces for Blockchain Copyright System" (http://www.musicbusinessworldwide.com/ascap-prs-sacem-join-forces-blockchain-copyright-system/). Music Business Worldwide. 9 April 2017. Archived (https://web.archive.org/web/20170410050815/http://www.musicbusinessworldwide.com/ascap-prs-sacem-join-forces-blockchain-copyright-system/) from the original on 10 April 2017.
128. Burchardi, K.; Harle, N. (20 January 2018). "The blockchain will disrupt the music business and beyond" (https://www.wired.co.uk/article/blockchain-disrupting-music-mycelia). *Wired UK*. Archived (https://web.archive.org/web/20180508040911/https://www.wired.co.uk/article/blockchain-disrupting-music-mycelia) from the original on 8 May 2018. Retrieved 8 May 2018.
129. Bartlett, Jamie (6 September 2015). "Imogen Heap: saviour of the music industry?" (https://www.theguardian.com/music/2015/sep/06/imogen-heap-saviour-of-music-industry). *The Guardian*. Archived (https://web.archive.org/web/20160422213153/http://www.theguardian.com/music/2015/sep/06/imogen-heap-saviour-of-music-industry) from the original on 22 April 2016. Retrieved 18 June 2016.
130. Wang, Kevin; Safavi, Ali (29 October 2016). "Blockchain is empowering the future of insurance" (https://techcrunch.com/2016/10/29/blockchain-is-empowering-the-future-of-insurance/). *Tech Crunch*. AOL Inc. Archived (https://web.archive.org/web/20161107160418/https://techcrunch.com/2016/10/29/blockchain-is-empowering-the-future-of-insurance/) from the original on 7 November 2016. Retrieved 7 November 2016.
131. Gatteschi, Valentina; Lamberti, Fabrizio; Demartini, Claudio; Pranteda, Chiara; Santamaría, Víctor (20 February 2018). "Blockchain and Smart Contracts for Insurance: Is the Technology Mature Enough?" (https://doi.org/10.3390%2Ffi10020020). *Future Internet*. **10** (2): 20. doi:10.3390/fi10020020 (https://doi.org/10.3390%2Ffi10020020).
132. "Blockchain reaction: Tech companies plan for critical mass" (http://www.ey.com/Publication/vwLUAssets/ey-blockchain-reaction-tech-companies-plan-for-critical-mass/$FILE/ey-blockchain-reaction.pdf) (PDF). Ernst & Young. p. 5. Archived (https://web.archive.org/web/20161114001423/http://www.ey.com/Publication/vwLUAssets/ey-blockchain-reaction-tech-companies-plan-for-critical-mass/$FILE/ey-blockchain-reaction.pdf) (PDF) from the original on 14 November 2016. Retrieved 13 November 2016.
133. Carrie Smith. Blockchain Reaction: How library professionals are approaching blockchain technology and its potential impact. (https://americanlibrariesmagazine.org/2019/03/01/library-blockchain-reaction/) Archived (https://web.archive.org/web/20190912044927/https://americanlibrariesmagazine.org/2019/03/01/library-blockchain-reaction/) 12 September 2019 at the Wayback Machine *American Libraries* March 2019.
134. "IBM Blockchain based on Hyperledger Fabric from the Linux Foundation" (https://www.ibm.com/blockchain/hyperledger.html). *IBM.com*. 9 January 2018. Archived (https://web.archive.org/web/20171207035925/https://www.ibm.com/blockchain/hyperledger.html) from the original on 7 December 2017. Retrieved 18 January 2018.
135. Hyperledger (22 January 2019). "Announcing Hyperledger Grid, a new project to help build and deliver supply chain solutions!" (https://www.hyperledger.org/blog/2019/01/22/announcing-hyperledger-grid-a-new-project-to-help-build-and-deliver-supply-chain-solutions). Archived (https://web.archive.org/web/20190204125611/https://www.hyperledger.org/blog/2019/01/22/announcing-hyperledger-grid-a-new-project-to-help-build-and-deliver-supply-chain-solutions) from the original on 4 February 2019. Retrieved 8 March 2019.

136. Mearian, Lucas (23 January 2019). "Grid, a new project from the Linux Foundation, will offer developers tools to create supply chain-specific applications running atop distributed ledger technology" (https://www.computerworld.com/article/3336036/blockchain/linuxs-hyperledger-to-give-developers-supply-chain-building-blocks.html). *Computerworld*. Archived (https://web.archive.org/web/20190203225703/https://www.computerworld.com/article/3336036/blockchain/linuxs-hyperledger-to-give-developers-supply-chain-building-blocks.html) from the original on 3 February 2019. Retrieved 8 March 2019.
137. "Why J.P. Morgan Chase Is Building a Blockchain on Ethereum" (http://fortune.com/2016/10/04/jp-morgan-chase-blockchain-ethereum-quorum/). *Fortune*. Archived (https://web.archive.org/web/20170202033844/http://fortune.com/2016/10/04/jp-morgan-chase-blockchain-ethereum-quorum/) from the original on 2 February 2017. Retrieved 24 January 2017.
138. Andoni, Merlinda; Robu, Valentin; Flynn, David; Abram, Simone; Geach, Dale; Jenkins, David; McCallum, Peter; Peacock, Andrew (2019). "Blockchain technology in the energy sector: A systematic review of challenges and opportunities" (https://www.sciencedirect.com/science/article/pii/S1364032118307184). *Renewable and Sustainable Energy Reviews*. **100**: 143–174. doi:10.1016/j.rser.2018.10.014 (https://doi.org/10.1016%2Fj.rser.2018.10.014). S2CID 116422191 (https://api.semanticscholar.org/CorpusID:116422191). Archived (https://web.archive.org/web/20200622122351/https://www.sciencedirect.com/science/article/pii/S1364032118307184) from the original on 22 June 2020. Retrieved 7 June 2020.
139. "This Blockchain-Based Energy Platform Is Building A Peer-To-Peer Grid" (https://www.fastcompany.com/40479952/this-blockchain-based-energy-platform-is-building-a-peer-to-peer-grid). 16 October 2017. Archived (https://web.archive.org/web/20200607085107/https://www.fastcompany.com/40479952/this-blockchain-based-energy-platform-is-building-a-peer-to-peer-grid) from the original on 7 June 2020. Retrieved 7 June 2020.
140. "Blockchain-based microgrid gives power to consumers in New York" (https://www.newscientist.com/article/2079334-blockchain-based-microgrid-gives-power-to-consumers-in-new-york/). Archived (https://web.archive.org/web/20160322204730/https://www.newscientist.com/article/2079334-blockchain-based-microgrid-gives-power-to-consumers-in-new-york/) from the original on 22 March 2016. Retrieved 7 June 2020.
141. Ma, Jinhua; Lin, Shih-Ya; Chen, Xin; Sun, Hung-Min; Chen, Yeh-Cheng; Wang, Huaxiong (2020). "A Blockchain-Based Application System for Product Anti-Counterfeiting" (https://doi.org/10.1109%2FACCESS.2020.2972026). *IEEE Access*. **8**: 77642–77652. doi:10.1109/ACCESS.2020.2972026 (https://doi.org/10.1109%2FACCESS.2020.2972026). ISSN 2169-3536 (https://www.worldcat.org/issn/2169-3536). S2CID 214205788 (https://api.semanticscholar.org/CorpusID:214205788).
142. Alzahrani, Naif; Bulusu, Nirupama (15 June 2018). "Block-Supply Chain: A New Anti-Counterfeiting Supply Chain Using NFC and Blockchain". *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. CryBlock'18. Munich, Germany: Association for Computing Machinery: 30–35. doi:10.1145/3211933.3211939 (https://doi.org/10.1145%2F3211933.3211939). ISBN 978-1-4503-5838-5. S2CID 169188795 (https://api.semanticscholar.org/CorpusID:169188795).
143. Balagurusamy, V. S. K.; Cabral, C.; Coomaraswamy, S.; Delamarche, E.; Dillenberger, D. N.; Dittmann, G.; Friedman, D.; Gökçe, O.; Hinds, N.; Jelitto, J.; Kind, A. (1 March 2019). "Crypto anchors" (https://ieeexplore.ieee.org/document/8645638). *IBM Journal of Research and Development*. **63** (2/3): 4:1–4:12. doi:10.1147/JRD.2019.2900651 (https://doi.org/10.1147%2FJRD.2019.2900651). ISSN 0018-8646 (https://www.worldcat.org/issn/0018-8646). S2CID 201109790 (https://api.semanticscholar.org/CorpusID:201109790).
144. Brett, Charles (18 April 2018). "EUIPO Blockathon Challenge 2018 -" (https://www.enterprisetimes.co.uk/2018/04/18/euipo-blockathon-challenge-2018/). *Enterprise Times*. Retrieved 1 September 2020.
145. "EUIPO Anti-Counterfeiting Blockathon Forum" (https://euipo.europa.eu/ohimportal/en/web/observatory/blockathon).

146. "PT Industrieel Management" (https://privacy.vakmedianet.nl/ptindustrieelmanagement/?ref=https%3A%2F%2Fwww.ptindustrieelmanagement.nl%2Fhse%2Fnieuws%2F2020%2F06%2Fnen-innoveert-certificatencheck-en-blockchain-1013171?_ga%3D2.11339894.336094574.1598975931-710406553.1598975931). *PT Industrieel Management*. Retrieved 1 September 2020.
147. "China selects pilot zones, application areas for blockchain project" (https://www.reuters.com/world/china/china-selects-pilot-zones-application-areas-blockchain-project-2022-01-30/). *Reuters*. 31 January 2022.
148. Wegner, Peter (March 1996). "Interoperability" (https://dl.acm.org/doi/10.1145/234313.234424). *ACM Computing Surveys*. **28**: 285–287. doi:10.1145/234313.234424 (https://doi.org/10.1145%2F234313.234424). Retrieved 24 October 2020.
149. Belchior, Rafael; Vasconcelos, André; Guerreiro, Sérgio; Correia, Miguel (May 2020). "A Survey on Blockchain Interoperability: Past, Present, and Future Trends". arXiv:2005.14282 (https://arxiv.org/abs/2005.14282) [cs.DC (https://arxiv.org/archive/cs.DC)].
150. Hardjono, T.; Hargreaves, M.; Smith, N. (2 October 2020). *An Interoperability Architecture for Blockchain Gateways* (Technical report). IETF. draft-hardjono-blockchain-interop-arch-00.
151. Hyun Song Shin (June 2018). "Chapter V. Cryptocurrencies: looking beyond the hype" (https://www.bis.org/publ/arpdf/ar2018e5.pdf) (PDF). *BIS 2018 Annual Economic Report*. Bank for International Settlements. Archived (https://web.archive.org/web/20180618080358/https://www.bis.org/publ/arpdf/ar2018e5.pdf) (PDF) from the original on 18 June 2018. Retrieved 19 June 2018. "Put in the simplest terms, the quest for decentralised trust has quickly become an environmental disaster."
152. Janda, Michael (18 June 2018). "Cryptocurrencies like bitcoin cannot replace money, says Bank for International Settlements" (http://www.abc.net.au/news/2018-06-18/cryptocurrencies-cannot-replace-money-bis/9879448). ABC (Australia). Archived (https://web.archive.org/web/20180618072225/http://www.abc.net.au/news/2018-06-18/cryptocurrencies-cannot-replace-money-bis/9879448) from the original on 18 June 2018. Retrieved 18 June 2018.
153. Hiltzik, Michael (18 June 2018). "Is this scathing report the death knell for bitcoin?" (https://latimes.com/business/hiltzik/la-fi-hiltzik-bitcoin-bank-20180618-story.html). *Los Angeles Times*. Archived (https://web.archive.org/web/20180618233532/http://www.latimes.com/business/hiltzik/la-fi-hiltzik-bitcoin-bank-20180618-story.html) from the original on 18 June 2018. Retrieved 19 June 2018.
154. Ossinger, Joanna (2 February 2022). "Polkadot Has Least Carbon Footprint, Crypto Researcher Says" (https://www.bloomberg.com/news/articles/2022-02-02/polkadot-has-smallest-carbon-footprint-crypto-researcher-says). *Bloomberg*. Retrieved 1 June 2022.
155. Jones, Jonathan Spencer (13 September 2021). "Blockchain proof-of-stake – not all are equal" (https://www.smart-energy.com/industry-sectors/new-technology/proof-of-stake-blockchains-not-all-are-equal/). *www.smart-energy.com*.
156. Criddle, Christina (February 20, 2021) "Bitcoin consumes 'more electricity than Argentina'." (https://www.bbc.com/news/technology-56012952) BBC News. (Retrieved April 26, 2021.)
157. Ponciano, Jonathan (March 9, 2021) "Bill Gates Sounds Alarm On Bitcoin's Energy Consumption–Here's Why Crypto Is Bad For Climate Change." (https://www.forbes.com/sites/jonathanponciano/2021/03/09/bill-gates-bitcoin-crypto-climate-change/) Forbes.com. (Retrieved April 26, 2021.)
158. Rowlatt, Justin (February 27, 2021) "How Bitcoin's vast energy use could burst its bubble." (https://www.bbc.com/news/science-environment-56215787) BBC News. (Retrieved April 26, 2021.)
159. Sorkin, Andrew et al. (March 9, 2021) "Why Bill Gates Is Worried About Bitcoin." (https://www.nytimes.com/2021/03/09/business/dealbook/bill-gates-bitcoin.html) *New York Times*. (Retrieved April 25, 2021.)

160. Illing, Sean (11 April 2018). "Why Bitcoin is bullshit, explained by an expert" (https://www.vox.com/conversations/2018/4/11/17206018/bitcoin-blockchain-cryptocurrency-weaver). *Vox*. Archived (https://web.archive.org/web/20180717041640/https://www.vox.com/conversations/2018/4/11/17206018/bitcoin-blockchain-cryptocurrency-weaver) from the original on 17 July 2018. Retrieved 17 July 2018.
161. Weaver, Nicholas. "Blockchains and Cryptocurrencies: Burn It With Fire" (https://www.youtube.com/watch?v=xCHab0dNnj4). *YouTube video*. Berkeley School of Information. Archived (https://web.archive.org/web/20190219015620/https://www.youtube.com/watch?v=xCHab0dNnj4) from the original on 19 February 2019. Retrieved 17 July 2018.
162. Köhler, Susanne; Pizzol, Massimo (20 November 2019). "Life Cycle Assessment of Bitcoin Mining" (https://doi.org/10.1021%2Facs.est.9b05687). *Environmental Science & Technology*. **53** (23): 13598–13606. Bibcode:2019EnST...5313598K (https://ui.adsabs.harvard.edu/abs/2019EnST...5313598K). doi:10.1021/acs.est.9b05687 (https://doi.org/10.1021%2Facs.est.9b05687). PMID 31746188 (https://pubmed.ncbi.nlm.nih.gov/31746188).
163. Stoll, Christian; Klaaßen, Lena; Gallersdörfer, Ulrich (2019). "The Carbon Footprint of Bitcoin" (https://doi.org/10.1016%2Fj.joule.2019.05.012). *Joule*. **3** (7): 1647–1661. doi:10.1016/j.joule.2019.05.012 (https://doi.org/10.1016%2Fj.joule.2019.05.012).
164. "US lawmakers begin probe into Bitcoin miners' high energy use" (https://www.business-standard.com/article/international/us-lawmakers-begin-probe-into-bitcoin-miners-high-energy-use-122012900282_1.html#:~:text=Eight+US+lawmakers+have+come,being+felt+across+the+globe). *Business Standard India*. 29 January 2022 – via Business Standard.
165. Cuen, Leigh (March 21, 2021) "The debate about cryptocurrency and data consumption." (https://techcrunch.com/2021/03/21/the-debate-about-cryptocurrency-and-energy-consumption/) TechCrunch. (Retrieved April 26, 2021.)
166. Catalini, Christian; Tucker, Catherine E. (11 August 2016). "Seeding the S-Curve? The Role of Early Adopters in Diffusion" (http://www.netinst.org/Catalini_16-02.pdf) (PDF). *SSRN*. doi:10.2139/ssrn.2822729 (https://doi.org/10.2139%2Fssrn.2822729). S2CID 157317501 (https://api.semanticscholar.org/CorpusID:157317501). SSRN 2822729 (https://ssrn.com/abstract=2822729).
167. Arnold, M. (2017) "Universities add blockchain to course list", Financial Times: Masters in Finance, Retrieved 26 January 2022. (https://www.ft.com/content/f736b04e-3708-11e7-99bd-13beb0903fa3)
168. Janssen, Marijn; Weerakkody, Vishanth; Ismagilova, Elvira; Sivarajah, Uthayasankar; Irani, Zahir (2020). "A framework for analysing blockchain technology adoption: Integrating institutional, market and technical factors" (https://doi.org/10.1016%2Fj.ijinfomgt.2019.08.012). *International Journal of Information Management*. Elsevier. **50**: 302–309. doi:10.1016/j.ijinfomgt.2019.08.012 (https://doi.org/10.1016%2Fj.ijinfomgt.2019.08.012).
169. Koens, Tommy; Poll, Erik (2019), "The Drivers Behind Blockchain Adoption: The Rationality of Irrational Choices", *Euro-Par 2018: Parallel Processing Workshops*, Lecture Notes in Computer Science, vol. 11339, pp. 535–546, doi:10.1007/978-3-030-10549-5_42 (https://doi.org/10.1007%2F978-3-030-10549-5_42), hdl:2066/200787 (https://hdl.handle.net/2066%2F200787), ISBN 978-3-030-10548-8, S2CID 57662305 (https://api.semanticscholar.org/CorpusID:57662305)
170. Li, Jerry (7 April 2020). "Blockchain Technology Adoption: Examining the Fundamental Drivers" (https://web.archive.org/web/20200605023137/https://dl.acm.org/doi/abs/10.1145/3396743.3396750). *Proceedings of the 2020 2nd International Conference on Management Science and Industrial Engineering*. Association for Computing Machinery: 253–260. doi:10.1145/3396743.3396750 (https://doi.org/10.1145%2F3396743.3396750). S2CID 218982506 (https://api.semanticscholar.org/CorpusID:218982506). Archived from the original (https://doi.org/10.1145/3396743.3396750) on 5 June 2020 – via ACM Digital Library.

171. Hsieh, Ying-Ying; Vergne, Jean-Philippe; Anderson, Philip; Lakhani, Karim; Reitzig, Markus (12 February 2019). "Correction to: Bitcoin and the rise of decentralized autonomous organizations" (https://doi.org/10.1186%2Fs41469-019-0041-1). *Journal of Organization Design*. **8** (1): 3. doi:10.1186/s41469-019-0041-1 (https://doi.org/10.1186%2Fs41469-019-0041-1). ISSN 2245-408X (https://www.worldcat.org/issn/2245-408X).
172. Felin, Teppo; Lakhani, Karim (2018). "What Problems Will You Solve With Blockchain?". *MIT Sloan Management Review*.
173. Beck, Roman; Mueller-Bloch, Christoph; King, John Leslie (2018). "Governance in the Blockchain Economy: A Framework and Research Agenda" (https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1835&context=jais). *Journal of the Association for Information Systems*: 1020–1034. doi:10.17705/1jais.00518 (https://doi.org/10.17705%2F1jais.00518). S2CID 69365923 (https://api.semanticscholar.org/CorpusID:69365923).
174. Popper, Nathaniel (27 June 2018). "What is the Blockchain? Explaining the Tech Behind Cryptocurrencies (Published 2018)" (https://www.nytimes.com/2018/06/27/business/dealbook/blockchains-guide-information.html). *The New York Times*.
175. Hugh Rooney, Brian Aiken, & Megan Rooney. (2017). Q&A. Is Internal Audit Ready for Blockchain? *Technology Innovation Management Review*, (10), 41.
176. Richard C. Kloch, Jr Simon J. Little, *Blockchain and Internal Audit* Internal Audit Foundation, 2019 ISBN 978-1-63454-065-0
177. Alexander, A. (2019). The audit, transformed: New advancements in technology are reshaping this core service. *Accounting Today*, 33(1)
178. Extance, Andy (30 September 2015). "The future of cryptocurrencies: Bitcoin and beyond" (https://doi.org/10.1038%2F526021a). *Nature*. **526** (7571): 21–23. Bibcode:2015Natur.526...21E (https://ui.adsabs.harvard.edu/abs/2015Natur.526...21E). doi:10.1038/526021a (https://doi.org/10.1038%2F526021a). ISSN 0028-0836 (https://www.worldcat.org/issn/0028-0836). OCLC 421716612 (https://www.worldcat.org/oclc/421716612). PMID 26432223 (https://pubmed.ncbi.nlm.nih.gov/26432223).
179. *Ledger (eJournal / eMagazine, 2015)*. OCLC. OCLC 910895894 (https://www.worldcat.org/oclc/910895894).
180. Hertig, Alyssa (15 September 2015). "Introducing Ledger, the First Bitcoin-Only Academic Journal" (http://motherboard.vice.com/read/introducing-ledger-the-first-bitcoin-only-academic-journal). *Motherboard*. Archived (https://web.archive.org/web/20170110172807/http://motherboard.vice.com/read/introducing-ledger-the-first-bitcoin-only-academic-journal) from the original on 10 January 2017. Retrieved 10 January 2017.
181. Rizun, Peter R.; Wilmer, Christopher E.; Burley, Richard Ford; Miller, Andrew (2015). "How to Write and Format an Article for Ledger" (http://ledger.pitt.edu/ojs/public/journals/1/AuthorGuide.pdf) (PDF). *Ledger*. **1** (1): 1–12. doi:10.5195/LEDGER.2015.1 (https://doi.org/10.5195%2FLEDGER.2015.1) (inactive 31 July 2022). ISSN 2379-5980 (https://www.worldcat.org/issn/2379-5980). OCLC 910895894 (https://www.worldcat.org/oclc/910895894). Archived (https://web.archive.org/web/20150922190603/http://ledger.pitt.edu/ojs/public/journals/1/AuthorGuide.pdf) (PDF) from the original on 22 September 2015. Retrieved 11 January 2017.

Further reading

- Crosby, Michael; Nachiappan; Pattanayak, Pradhan; Verma, Sanjeev; Kalyanaraman, Vignesh (16 October 2015). BlockChain Technology: Beyond Bitcoin (http://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf) (PDF) (Report). Sutardja Center for Entrepreneurship & Technology Technical Report. University of California, Berkeley. Retrieved 19 March 2017.
- Jaikaran, Chris (28 February 2018). *Blockchain: Background and Policy Issues* (https://crsreports.congress.gov/product/pdf/R/R45116/3). Washington, DC: Congressional Research Service. Retrieved 2 December 2018.

- Kakavand, Hossein; De Sevres, Nicolette Kost; Chilton, Bart (12 October 2016). The Blockchain Revolution: An Analysis of Regulation and Technology Related to Distributed Ledger Technologies (Report). Luther Systems & DLA Piper. SSRN 2849251 (https://ssrn.com/abstract=2849251).
- Mazonka, Oleg (29 December 2016). "Blockchain: Simple Explanation" (http://jrxv.net/x/16/chain.pdf) (PDF). *Journal of Reference*.
- Tapscott, Don; Tapscott, Alex (2016). *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business and the World*. London: Portfolio Penguin. ISBN 978-0-241-23785-4. OCLC 971395169 (https://www.worldcat.org/oclc/971395169).
- Saito, Kenji; Yamada, Hiroyuki (June 2016). *What's So Different about Blockchain? Blockchain is a Probabilistic State Machine*. IEEE 36th International Conference on Distributed Computing Systems Workshops. *International Conference on Distributed Computing Systems Workshops (Icdcs)*. Nara, Nara, Japan: IEEE. pp. 168–75. doi:10.1109/ICDCSW.2016.28 (https://doi.org/10.1109%2FICDCSW.2016.28). ISBN 978-1-5090-3686-8. ISSN 2332-5666 (https://www.worldcat.org/issn/2332-5666).
- Raval, Siraj (2016). *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology* (http://shop.oreilly.com/product/0636920039334.do?sortby=publicationDate). Oreilly. ISBN 9781491924549.
- Bashir, Imran (2017). *Mastering Blockchain*. Packt Publishing, Ltd. ISBN 978-1-78712-544-5. OCLC 967373845 (https://www.worldcat.org/oclc/967373845).
- Knirsch, Fabian; Unterweger, Andread; Engel, Dominik (2019). "Implementing a blockchain from scratch: why, how, and what we learned" (https://jis-eurasipjournals.springeropen.com/articles/10.1186/s13635-019-0085-3#citeas). *EURASIP Journal on Information Security*. **2019**. doi:10.1186/s13635-019-0085-3 (https://doi.org/10.1186%2Fs13635-019-0085-3). S2CID 84837476 (https://api.semanticscholar.org/CorpusID:84837476).
- D. Puthal, N. Malik, S. P. Mohanty, E. Kougianos, and G. Das, "Everything you Wanted to Know about the Blockchain (http://www.smohanty.org/Publications_Journals/2018/Mohanty_IEEE-CEM_2018-Jul_Blockchain.pdf)", *IEEE Consumer Electronics Magazine*, Volume 7, Issue 4, July 2018, pp. 06–14.
- David L. Portilla, David J. Kappos, Minh Van Ngo, Sasha Rosenthal-Larrea, John D. Buretta and Christopher K. Fargo, Cravath, Swaine & Moore LLP, "Blockchain in the Banking Sector: A Review of the Landscape and Opportunities (https://corpgov.law.harvard.edu/2022/01/28/blockchain-in-the-banking-sector-a-review-of-the-landscape-and-opportunities)", *Harvard Law School of Corporate Governance*, posted on Friday, January 28, 2022

External links

- Media related to Blockchain at Wikimedia Commons

Distributed ledger

A **distributed ledger** (also called a **shared ledger** or **distributed ledger technology** or **DLT**) is the consensus of replicated, shared, and synchronized digital data that is geographically spread (distributed) across many sites, countries, or institutions.[1] In contrast to a centralized database, a distributed ledger does not require a central administrator, and consequently does not have a single (central) point-of-failure.[2][3][4]

In general, a distributed ledger requires a peer-to-peer (P2P) computer network and consensus algorithms so that the ledger is reliably replicated across distributed computer nodes (servers, clients, etc.).[2] The most common form of distributed ledger technology is the blockchain (commonly associated with the Bitcoin cryptocurrency), which can either be on a public or private network.[3] Infrastructure for data management is a common barrier to implementing DLT. [5]

In some cases, where the distributed digital information functions as an accounting journal rather than an accounting ledger, another term is used: **RJT** for r**eplicated journal technology**.[6]

Contents

Characteristics

Distributed ledger data is typically spread across multiple nodes (computational devices) on a P2P network, where each replicates and saves an identical copy of the ledger data and updates itself independently of other nodes. The primary advantage of this distributed processing pattern is the lack of a central authority, which would constitute a single point of failure. When a ledger update transaction is broadcast to the P2P network, each distributed node processes a new update transaction independently, and then collectively all working nodes use a consensus algorithm to determine the correct copy of the updated ledger. Once a consensus has been determined, all the other nodes update themselves with the latest, correct copy of the updated ledger.[7][8] Security is enforced through cryptographic keys and signatures.[9][10][11]

Applications

In 2016, some banks tested distributed ledger systems for payments[12] to determine their usefulness.[2] In 2020, Axoni launched Veris, a distributed ledger platform that manages equity swap transactions.[13] The platform, which matches and reconciles post-trade data on stock swaps, is used by BlackRock Inc., Goldman Sachs Group Inc., and Citigroup, Inc.[14]

Types

Distributed ledger technologies can be categorized in terms of their data structures, consensus algorithms, permissions, and whether they are mined.[15] DLT data structure types include linear data structures (blockchains) to more complex directed acyclic graph (DAG) and hybrid data structures. DLT consensus algorithm types includeproof-of-work (PoW) and proof-of-stake (PoS) algorithms and DAG consensus-building and voting algorithms. DLTs are generally either permissioned (public) or permissionless (private).[16] PoW cryptocurrencies are generally either 'mined' or 'non-mined', where the latter typically indicates 'pre-mined' cryptocurrencies, such as XRP or IOTA.[17] PoS cryptocurrencies do not use miners, instead usually relying on validation among owners of the cryptocurrency, such as Cardano or Solana.

Blockchains are the most common DLT type, with a 256-bit secure hash algorithm (SHA). DLTs based on DAG data structures or hybrid blockchain-DAG decrease transaction data size and transaction costs, while increasing transaction speeds compared with Bitcoin, the first cryptocurrency.[18][19][20] Examples of DAG DLT cryptocurrencies include MIOTA (IOTA Tangle DLT) and HBAR (Hedera Hashgraph,a patented DLT).

	Blockchain	Tangle	Hashgraph
Technology	Blockchain	Directed acyclic graph	Directed acyclic graph
Copyright	Open source	Open source	Patented
Consensus	Proof of work (PoW): SHA-256	Proof-of-work (PoW): Check of tangle tip	Virtual voting
	Proof of stake (PoS): SHA-256		
Openness	Public ledger	Public ledger	Private ledger
Applications	Bitcoin, various altcoin DLTs	IOTA	Hedera Hashgraph
Efficiency (Tx/sec; TPS)	Relatively low TPS	Relatively high TPS	Extremely high TPS

See also

- Hyperledger
- Decentralized Finance (DeFi)
- Cryptoeconomics
- Eventual consistency
- Web3

References

1. Distributed Ledger Technology: beyond block chain (https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distributed-ledger-technology.pdf) (PDF) (Report). Government Office for Science (UK). January 2016. Retrieved 29 August 2016.
2. Scardovi, Claudio (2016). *Restructuring and Innovation in Banking* (https://play.google.com/store/books/details?id=uNM0DQAAQBAJ). Springer. p. 36. ISBN 978-331940204-8. Retrieved 21 November 2016.
3. "Crypto FAQ: What is Distributed Ledger Technology (DLT)?" (https://cryptocurrencyworks.com/faq//labels//faq/what-is-distributed-ledger-tech.html). *CryptoCurrency Works™*. Retrieved 2022-08-09.
4. "Distributed Ledgers" (https://www.investopedia.com/terms/d/distributed-ledgers.asp). *Investopedia*. Retrieved 2022-08-09.

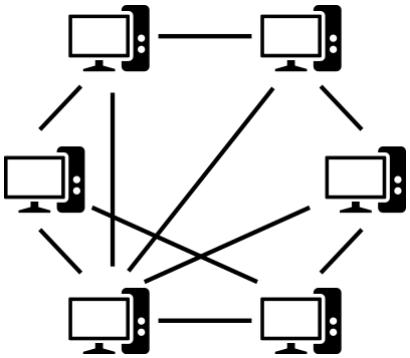
5. Sadeghi, Mahsa; Mahmoudi, Amin; Deng, Xiaopeng (2022-02-01). "Adopting distributed ledger technology for the sustainable construction industry: evaluating the barriers using Ordinal Priority Approach" (https://doi.org/10.1007/s11356-021-16376-y). *Environmental Science and Pollution Research*. **29** (7): 10495–10520. doi:10.1007/s11356-021-16376-y (https://doi.org/10.1007%2Fs11356-021-16376-y). ISSN 1614-7499 (https://www.worldcat.org/issn/1614-7499). PMC 8443118 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8443118). PMID 34528198 (https://pubmed.ncbi.nlm.nih.gov/34528198).
6. S, Surbhi (26 Jul 2018). "Difference Between Journal and Ledger" (https://keydifferences.com/difference-between-journal-and-ledger.html). *Developer works*. Retrieved 22 Dec 2020.
7. Maull, Roger; Godsiff, Phil; Mulligan, Catherine; Brown, Alan; Kewell, Beth (21 Sep 2017). "Distributed ledger technology: Applications and implications" (https://openresearch.surrey.ac.uk/esploro/outputs/journalArticle/Distributed-Ledger-Technology-Applications-and-Implications/99514404202346). *FINRA*. **26** (5): 481–89. doi:10.1002/jsc.2148 (https://doi.org/10.1002%2Fjsc.2148).
8. Ray, Shaan (2018-02-20). "The Difference Between Blockchains & Distributed Ledger Technology" (https://towardsdatascience.com/the-difference-between-blockchains-distributed-ledger-technology-42715a0fa92). *Towards Data Science*. Retrieved 25 September 2018.
9. "Distributed Ledger Technology: beyond block chain" (https://www.gov.uk/government/news/distributed-ledger-technology-beyond-block-chain) (Press release). Government Office for Science (UK). 19 January 2016. Retrieved 25 September 2018.
10. Brakeville, Sloane; Perepa, Bhargav (18 Mar 2018). "Blockchain basics: Introduction to distributed ledgers" (https://www.ibm.com/developerworks/cloud/library/cl-blockchain-basics-intro-bluemix-trs/). *Developer works*. IBM. Retrieved 25 Sep 2018.
11. Rutland, Emily. "Blockchain Byte" (https://www.finra.org/sites/default/files/2017_BC_Byte.pdf) (PDF). *FINRA*. R3 Research. p. 2. Retrieved 25 September 2018.
12. "Central banks look to the future of money with blockchain technology trial" (http://www.afr.com/technology/central-banks-look-to-the-future-of-money-with-blockchain-technology-trial-20161117-gss4nd). *Australian Financial Review*. Fairfax Media Publications. 21 November 2016. Retrieved 7 December 2016.
13. "Citi and Goldman Sachs go live with blockchain equity swaps platform - The TRADE". www.thetradenews.com. Retrieved 2022-05-20.
14. "BlackRock Joins Blockchain Platform Axoni for Equity Swap Trades". Bloomberg.com. 2021-09-07. Retrieved 2022-05-20.
15. "Crypto FAQ: What is Distributed Ledger Technology (DLT)?" (https://cryptocurrencyworks.com/crypto-faq/what-is-distributed-ledger-tech.html). *Cryptocurrency Works*. Retrieved 2022-07-28.
16. "Blockchains & Distributed Ledger Technologies" (https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/). *Blockchainhub Berlin*. Retrieved 2022-07-28.
17. "Crypto FAQ: What is Distributed Ledger Technology (DLT)?" (https://cryptocurrencyworks.com/crypto-faq/what-is-distributed-ledger-tech.html). *Cryptocurrency Works*. Retrieved 2022-07-28.
18. Schueffel, Patrick (2017-12-15). "Alternative Distributed Ledger Technologies Blockchain vs. Tangle vs. Hashgraph - A High-Level Overview and Comparison" (https://papers.ssrn.com/abstract=3144241). Rochester, NY. SSRN 3144241 (https://ssrn.com/abstract=3144241).
19. Pervez, H. (2018). "A Comparative Analysis of DAG-Based Blockchain Architectures" (https://www.researchgate.net/publication/330880551_A_Comparative_Analysis_of_DAG-Based_Blockchain_Architectures). ICOSST 2018.
20. "Crypto FAQ: What is Distributed Ledger Technology (DLT)?" (https://cryptocurrencyworks.com/crypto-faq/what-is-distributed-ledger-tech.html). *Cryptocurrency Works*. Retrieved 2022-07-28.

Peer-to-peer

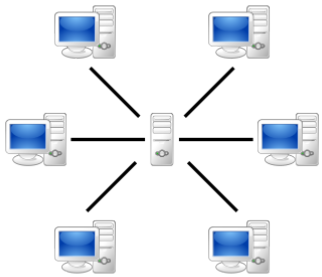
Peer-to-peer (**P2P**) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the network. They are said to form a peer-to-peer network of nodes.[1]

Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts.[2] Peers are both suppliers and consumers of resources, in contrast to the traditional client–server model in which the consumption and supply of resources is divided.[3]

While P2P systems had previously been used in many application domains,[4] the architecture was popularized by the file sharing system Napster, originally released in 1999.[5] The concept has inspired new structures and philosophies in many areas of human interaction. In such social contexts, peer-to-peer as a meme refers to the egalitarian social networking that has emerged throughout society, enabled by Internet technologies in general.



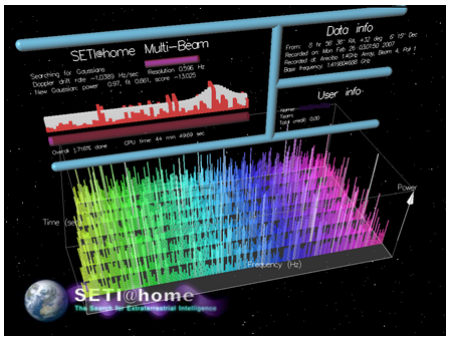
A **peer-to-peer (P2P) network** in which interconnected nodes ("peers") share resources amongst each other without the use of a centralized administrative system



A network based on the **client–server model**, where individual *clients* request services and resources from centralized servers

Contents

Historical development



SETI@home was established in 1999

While P2P systems had previously been used in many application domains,[4] the concept was popularized by file sharing systems such as the music-sharing application Napster (originally released in 1999). The peer-to-peer movement allowed millions of Internet users to connect "directly, forming groups and collaborating to become user-created search engines, virtual supercomputers, and filesystems."[6] The basic concept of peer-to-peer computing was envisioned in earlier software systems and networking discussions, reaching back to principles stated in the first Request for Comments, RFC 1.[7]

Tim Berners-Lee's vision for the World Wide Web was close to a P2P network in that it assumed each user of the web would be an active editor and contributor, creating and linking content to form an interlinked "web" of links. The early Internet was more open than present day, where two machines connected to the Internet could send packets to each other without firewalls and other security measures.[6] This contrasts to the broadcasting-like structure of the web as it has developed over the years.[8][9][10] As a precursor to the Internet, ARPANET was a successful client–server network where "every participating node could request and serve content." However, ARPANET was not self-organized, and it lacked the ability to "provide any means for context or content-based routing beyond 'simple' address-based routing."[10]

Therefore, Usenet, a distributed messaging system that is often described as an early peer-to-peer architecture, was established. It was developed in 1979 as a system that enforces a decentralized model of control.[11] The basic model is a client–server model from the user or client perspective that offers a self-organizing approach to newsgroup servers. However, news servers communicate with one another as peers to propagate Usenet news articles over the entire group of network servers. The same consideration applies to SMTP email in the sense that the core email-relaying network of mail transfer agents has a peer-to-peer character, while the periphery of Email clients and their direct connections is strictly a client–server relationship.

In May 1999, with millions more people on the Internet, Shawn Fanning introduced the music and file-sharing application called Napster.[10] Napster was the beginning of peer-to-peer networks, as we know them today, where "participating users establish a virtual network, entirely independent from the physical network, without having to obey any administrative authorities or restrictions."[10]

Architecture

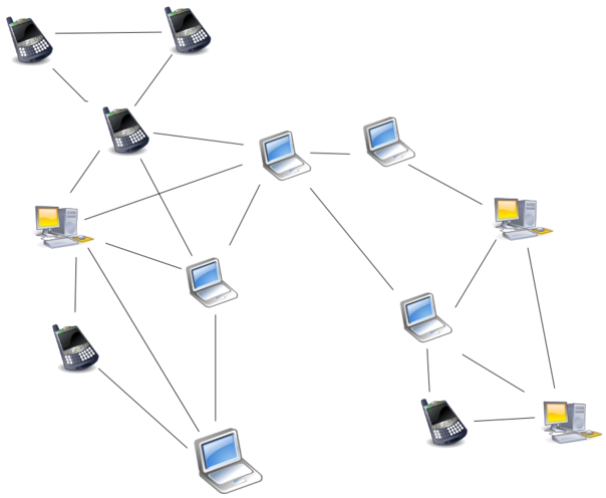
A peer-to-peer network is designed around the notion of equal *peer* nodes simultaneously functioning as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client–server model where communication is usually to and from a central server. A typical example of a file transfer that uses the client–server model is the File Transfer Protocol (FTP) service in which the client and server programs are distinct: the clients initiate the transfer, and the servers satisfy these requests.

Routing and resource discovery

Peer-to-peer networks generally implement some form of virtual overlay network on top of the physical network topology, where the nodes in the overlay form a subset of the nodes in the physical network. Data is still exchanged directly over the underlying TCP/IP network, but at the application layer peers are able to communicate with each other directly, via the logical overlay links (each of which corresponds to a path through the underlying physical network). Overlays are used for indexing and peer discovery, and make the P2P system independent from the physical network topology. Based on how the nodes are linked to each other within the overlay network, and how resources are indexed and located, we can classify networks as *unstructured* or *structured* (or as a hybrid between the two).[12][13][14]

Unstructured networks

Unstructured peer-to-peer networks do not impose a particular structure on the overlay network by design, but rather are formed by nodes that randomly form connections to each other.[15] (Gnutella, Gossip, and Kazaa are examples of unstructured P2P protocols).[16]



Overlay network diagram for an **unstructured P2P network**, illustrating the ad hoc nature of the connections between nodes

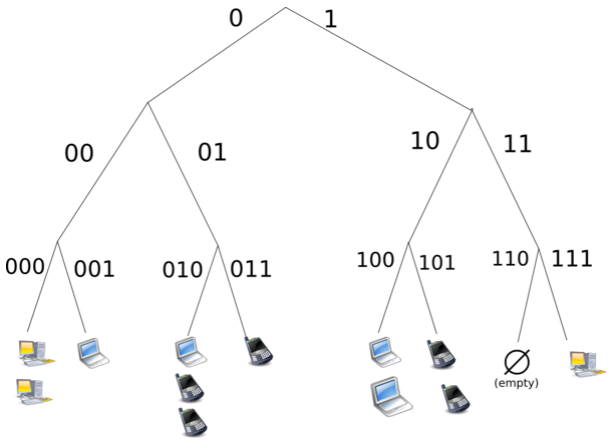
Because there is no structure globally imposed upon them, unstructured networks are easy to build and allow for localized optimizations to different regions of the overlay.[17] Also, because the role of all peers in the network is the same, unstructured networks are highly robust in the face of high rates of "churn"—that is, when large numbers of peers are frequently joining and leaving the network.[18][19]

However, the primary limitations of unstructured networks also arise from this lack of structure. In particular, when a peer wants to find a desired piece of data in the network, the search query must be flooded through the network to find as many peers as possible that share the data. Flooding causes a very high amount of signaling traffic in the network, uses more CPU/memory (by requiring every peer to process all search queries), and does not ensure that search queries will always be resolved. Furthermore, since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired data. Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing. But if a peer is looking for rare data shared by only a few other peers, then it is highly unlikely that search will be successful.[20]

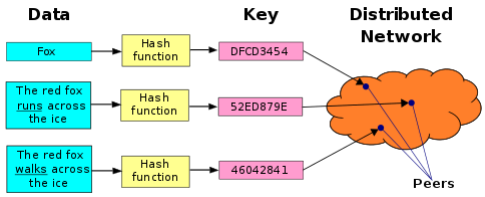
Structured networks

In *structured peer-to-peer networks* the overlay is organized into a specific topology, and the protocol ensures that any node can efficiently[21] search the network for a file/resource, even if the resource is extremely rare.

The most common type of structured P2P networks implement a distributed hash table (DHT),[22][23] in which a variant of consistent hashing is used to assign ownership of each file to a particular peer.[24][25] This enables peers to search for resources on the network using a hash table: that is, (*key*, *value*) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key.[26][27]



Overlay network diagram for a **structured P2P network**, using a distributed hash table (DHT) to identify and locate nodes/resources



Distributed hash tables

However, in order to route traffic efficiently through the network, nodes in a structured overlay must maintain lists of neighbors[28] that satisfy specific criteria. This makes them less robust in networks with a high rate of *churn* (i.e. with large numbers of nodes frequently joining and leaving the network).[19][29] More recent evaluation of P2P resource discovery solutions under real workloads have pointed out several issues in DHT-based solutions such as high cost of advertising/discovering resources and static and dynamic load imbalance.[30]

Notable distributed networks that use DHTs include Tixati, an alternative to BitTorrent's distributed tracker, the Kad network, the Storm botnet, and the YaCy. Some prominent research projects include the Chord project, Kademlia, PAST storage utility, P-Grid, a self-organized and emerging overlay network, and CoopNet content distribution system.[31] DHT-based networks have also been widely utilized for accomplishing efficient resource discovery[32][33] for grid computing systems, as it aids in resource management and scheduling of applications.

Hybrid models

Hybrid models are a combination of peer-to-peer and client–server models.[34] A common hybrid model is to have a central server that helps peers find each other. Spotify was an example of a hybrid model [until 2014]. There are a variety of hybrid models, all of which make trade-offs between the centralized functionality provided by a structured server/client network and the node equality afforded by the pure peer-to-peer unstructured networks. Currently, hybrid models have better performance than either pure unstructured networks or pure structured networks because certain functions, such as searching, do require a centralized functionality but benefit from the decentralized aggregation of nodes provided by unstructured networks.[35]

CoopNet content distribution system

CoopNet (Cooperative Networking) was a proposed system for off-loading serving to peers who have recently downloaded content, proposed by computer scientists Venkata N. Padmanabhan and Kunwadee Sripanidkulchai, working at Microsoft Research and Carnegie Mellon University.[36][37] When a server experiences an increase in load it redirects incoming peers to other peers who have agreed to mirror the content, thus off-loading balance from the server. All of the information is retained at the server. This system makes use of the fact that the bottle-neck is

most likely in the outgoing bandwidth than the CPU, hence its server-centric design. It assigns peers to other peers who are 'close in IP' to its neighbors [same prefix range] in an attempt to use locality. If multiple peers are found with the same file it designates that the node choose the fastest of its neighbors. Streaming media is transmitted by having clients cache the previous stream, and then transmit it piece-wise to new nodes.

Security and trust

Peer-to-peer systems pose unique challenges from a computer security perspective.

Like any other form of software, P2P applications can contain vulnerabilities. What makes this particularly dangerous for P2P software, however, is that peer-to-peer applications act as servers as well as clients, meaning that they can be more vulnerable to remote exploits.[38]

Routing attacks

Since each node plays a role in routing traffic through the network, malicious users can perform a variety of "routing attacks", or denial of service attacks. Examples of common routing attacks include "incorrect lookup routing" whereby malicious nodes deliberately forward requests incorrectly or return false results, "incorrect routing updates" where malicious nodes corrupt the routing tables of neighboring nodes by sending them false information, and "incorrect routing network partition" where when new nodes are joining they bootstrap via a malicious node, which places the new node in a partition of the network that is populated by other malicious nodes.[39]

Corrupted data and malware

The prevalence of malware varies between different peer-to-peer protocols. Studies analyzing the spread of malware on P2P networks found, for example, that 63% of the answered download requests on the gnutella network contained some form of malware, whereas only 3% of the content on OpenFT contained malware. In both cases, the top three most common types of malware accounted for the large majority of cases (99% in gnutella, and 65% in OpenFT). Another study analyzing traffic on the Kazaa network found that 15% of the 500,000 file sample taken were infected by one or more of the 365 different computer viruses that were tested for.[40]

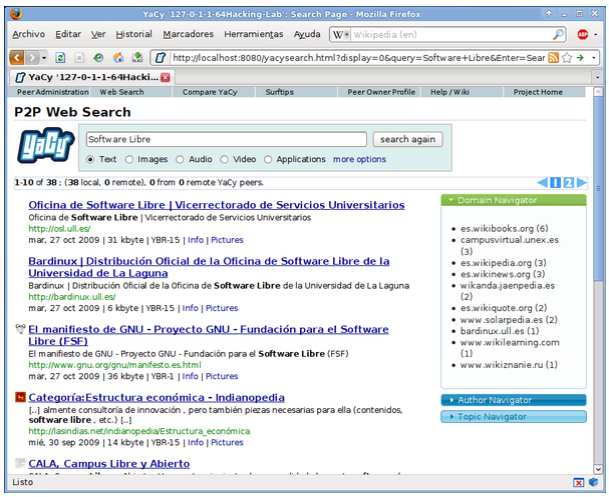
Corrupted data can also be distributed on P2P networks by modifying files that are already being shared on the network. For example, on the FastTrack network, the RIAA managed to introduce faked chunks into downloads and downloaded files (mostly MP3 files). Files infected with the RIAA virus were unusable afterwards and contained malicious code. The RIAA is also known to have uploaded fake music and movies to P2P networks in order to deter illegal file sharing.[41] Consequently, the P2P networks of today have seen an enormous increase of their security and file verification mechanisms. Modern hashing, chunk verification and different encryption methods have made most networks resistant to almost any type of attack, even when major parts of the respective network have been replaced by faked or nonfunctional hosts.[42]

Resilient and scalable computer networks

The decentralized nature of P2P networks increases robustness because it removes the single point of failure that can be inherent in a client–server based system.[43] As nodes arrive and demand on the system increases, the total capacity of the system also increases, and the likelihood of failure decreases. If one peer on the network fails to function properly, the whole network is not compromised or damaged. In contrast, in a typical client–server architecture, clients share only

their demands with the system, but not their resources. In this case, as more clients join the system, fewer resources are available to serve each client, and if the central server fails, the entire network is taken down.

Distributed storage and search



Search results for the query "software libre", using YaCy a free distributed search engine that runs on a peer-to-peer network instead making requests to centralized index servers (like Google, Yahoo, and other corporate search engines)

There are both advantages and disadvantages in P2P networks related to the topic of data backup, recovery, and availability. In a centralized network, the system administrators are the only forces controlling the availability of files being shared. If the administrators decide to no longer distribute a file, they simply have to remove it from their servers, and it will no longer be available to users. Along with leaving the users powerless in deciding what is distributed throughout the community, this makes the entire system vulnerable to threats and requests from the government and other large forces. For example, YouTube has been pressured by the RIAA, MPAA, and entertainment industry to filter out copyrighted content. Although server-client networks are able to monitor and manage content availability, they can have more stability in the availability of the content they choose to host. A client should not have trouble accessing obscure content that is being shared on a stable centralized network. P2P networks, however, are more unreliable in sharing unpopular files because sharing files in a P2P network requires that at least one node in the network has the requested data, and that node must be able to connect to the node requesting the data. This requirement is occasionally hard to meet because users may delete or stop sharing data at any point.[44]

In this sense, the community of users in a P2P network is completely responsible for deciding what content is available. Unpopular files will eventually disappear and become unavailable as more people stop sharing them. Popular files, however, will be highly and easily distributed. Popular files on a P2P network actually have more stability and availability than files on central networks. In a centralized network, a simple loss of connection between the server and clients is enough to cause a failure, but in P2P networks, the connections between every node must be lost in order to cause a data sharing failure. In a centralized system, the administrators are responsible for all data recovery and backups, while in P2P systems, each node requires its own backup system. Because of the lack of central authority in P2P networks, forces such as the recording industry, RIAA, MPAA, and the government are unable to delete or stop the sharing of content on P2P systems.[45]

Applications

Content delivery

In P2P networks, clients both provide and use resources. This means that unlike client–server systems, the content-serving capacity of peer-to-peer networks can actually *increase* as more users begin to access the content (especially with protocols such as Bittorrent that require users to share, refer a performance measurement study[46]). This property is one of the major advantages of using P2P networks because it makes the setup and running costs very small for the original content distributor.[47][48]

File-sharing networks

Many peer-to-peer file sharing networks, such as Gnutella, G2, and the eDonkey network popularized peer-to-peer technologies.

- Peer-to-peer content delivery networks.
- Peer-to-peer content services, e.g. caches for improved performance such as Correli Caches[49]
- Software publication and distribution (Linux distribution, several games); via file sharing networks.

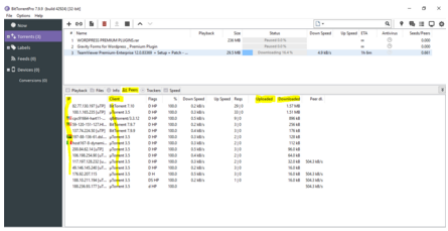
Copyright infringements

Peer-to-peer networking involves data transfer from one user to another without using an intermediate server. Companies developing P2P applications have been involved in numerous legal cases, primarily in the United States, over conflicts with copyright law.[50] Two major cases are *Grokster vs RIAA* and *MGM Studios, Inc. v. Grokster, Ltd.*.[51] In the last case, the Court unanimously held that defendant peer-to-peer file sharing companies Grokster and Streamcast could be sued for inducing copyright infringement.

Multimedia

- The P2PTV and PDTP protocols.
- Some proprietary multimedia applications use a peer-to-peer network along with streaming servers to stream audio and video to their clients.
- Peercasting for multicasting streams.
- Pennsylvania State University, MIT and Simon Fraser University are carrying on a project called LionShare designed for facilitating file sharing among educational institutions globally.
- Osiris is a program that allows its users to create anonymous and autonomous web portals distributed via P2P network.

Other P2P applications



Torrent file connect peers

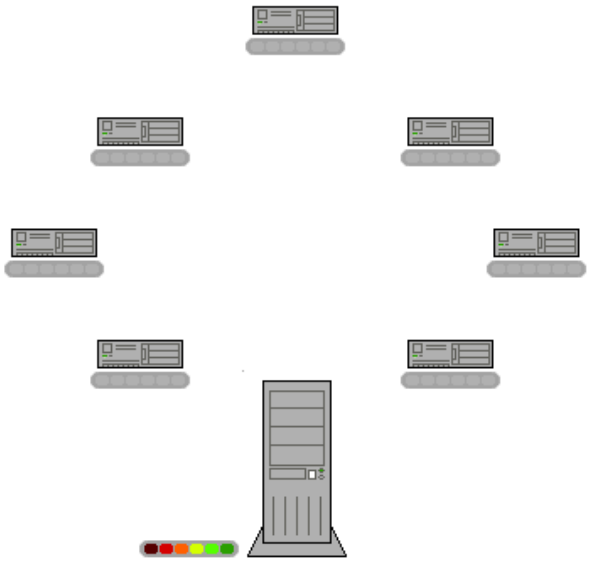
- Bitcoin and additional such as Ether, Nxt and Peercoin are peer-to-peer-based digital cryptocurrencies.
- Dat, a distributed version-controlled publishing platform.
- Filecoin is an open source, public, cryptocurrency and digital payment system intended to be a blockchain-based cooperative digital storage and data retrieval method.
- I2P, an overlay network used to browse the Internet anonymously.
- Unlike the related I2P, the Tor network is not itself peer-to-peer; however, it can enable peer-to-peer applications to be built on top of it via onion services.
- The InterPlanetary File System (IPFS) is a protocol and network designed to create a content-addressable, peer-to-peer method of storing and sharing hypermedia distribution protocol. Nodes in the IPFS network form a distributed file system.
- Jami, a peer-to-peer chat and SIP app.
- JXTA, a peer-to-peer protocol designed for the Java platform.
- Netsukuku, a Wireless community network designed to be independent from the Internet.
- Open Garden, connection sharing application that shares Internet access with other devices using Wi-Fi or Bluetooth.

- Resilio Sync, a directory-syncing app.
- Research like the Chord project, the PAST storage utility, the P-Grid, and the CoopNet content distribution system.
- Syncthing, a directory-syncing app.
- Tradepal and M-commerce applications that power real-time marketplaces.
- The U.S. Department of Defense is conducting research on P2P networks as part of its modern network warfare strategy.[52] In May, 2003, Anthony Tether, then director of DARPA, testified that the United States military uses P2P networks.
- WebTorrent is a P2P streaming torrent client in JavaScript for use in web browsers, as well as in the WebTorrent Desktop stand alone version that bridges WebTorrent and BitTorrent serverless networks.
- Microsoft in Windows 10 uses a proprietary peer to peer technology called "Delivery Optimization" to deploy operating system updates using end-users PCs either on the local network or other PCs. According to Microsoft's Channel 9 it led to a 30%-50% reduction in Internet bandwidth usage.[53]
- Artisoft's LANtastic was built as a peer-to-peer operating system. Machines can be both servers and workstations at the same time.
- Hotline Communications Hotline Client was built as decentralized servers with tracker software dedicated to any type of files and still operates today

Social implications

Incentivizing resource sharing and cooperation

Cooperation among a community of participants is key to the continued success of P2P systems aimed at casual human users; these reach their full potential only when large numbers of nodes contribute resources. But in current practice, P2P networks often contain large numbers of users who utilize resources shared by other nodes, but who do not share anything themselves (often referred to as the "freeloader problem"). Freeloading can have a profound impact on the network and in some cases can cause the community to collapse.[54] In these types of networks "users have natural disincentives to cooperate because cooperation consumes their own resources and may degrade their own performance."[55] Studying the social attributes of P2P networks is challenging due to large populations of turnover, asymmetry of interest and zero-cost identity.[55] A variety of incentive mechanisms have been implemented to encourage or even force nodes to contribute resources.[56]



The BitTorrent protocol: In this animation, the colored bars beneath all of the 7 clients in the upper region above represent the file being shared, with each color representing an individual piece of the file. After the initial pieces transfer from the seed (large system at the bottom), the pieces are individually transferred from client to client. The original seeder only needs to send out one copy of the file for all the clients to receive a copy.

Some researchers have explored the benefits of enabling virtual communities to self-organize and introduce incentives for resource sharing and cooperation, arguing that the social aspect missing from today's P2P systems should be seen both as a goal and a means for self-organized virtual communities to be built and fostered.[57] Ongoing

research efforts for designing effective incentive mechanisms in P2P systems, based on principles from game theory, are beginning to take on a more psychological and information-processing direction.

Privacy and anonymity

Some peer-to-peer networks (e.g. Freenet) place a heavy emphasis on privacy and anonymity—that is, ensuring that the contents of communications are hidden from eavesdroppers, and that the identities/locations of the participants are concealed. Public key cryptography can be used to provide encryption, data validation, authorization, and authentication for data/messages. Onion routing and other mix network protocols (e.g. Tarzan) can be used to provide anonymity.[58]

Perpetrators of live streaming sexual abuse and other cybercrimes have used peer-to-peer platforms to carry out activities with anonymity.[59]

Political implications

Intellectual property law and illegal sharing

Although peer-to-peer networks can be used for legitimate purposes, rights holders have targeted peer-to-peer over the involvement with sharing copyrighted material. Peer-to-peer networking involves data transfer from one user to another without using an intermediate server. Companies developing P2P applications have been involved in numerous legal cases, primarily in the United States, primarily over issues surrounding copyright law.[50] Two major cases are *Grokster vs RIAA* and *MGM Studios, Inc. v. Grokster, Ltd.*[51] In both of the cases the file sharing technology was ruled to be legal as long as the developers had no ability to prevent the sharing of the copyrighted material. To establish criminal liability for the copyright infringement on peer-to-peer systems, the government must prove that the defendant infringed a copyright willingly for the purpose of personal financial gain or commercial advantage.[60] Fair use exceptions allow limited use of copyrighted material to be downloaded without acquiring permission from the rights holders. These documents are usually news reporting or under the lines of research and scholarly work. Controversies have developed over the concern of illegitimate use of peer-to-peer networks regarding public safety and national security. When a file is downloaded through a peer-to-peer network, it is impossible to know who created the file or what users are connected to the network at a given time. Trustworthiness of sources is a potential security threat that can be seen with peer-to-peer systems.[61]

A study ordered by the European Union found that illegal downloading *may* lead to an increase in overall video game sales because newer games charge for extra features or levels. The paper concluded that piracy had a negative financial impact on movies, music, and literature. The study relied on self-reported data about game purchases and use of illegal download sites. Pains were taken to remove effects of false and misremembered responses.[62][63][64]

Network neutrality

Peer-to-peer applications present one of the core issues in the network neutrality controversy. Internet service providers (ISPs) have been known to throttle P2P file-sharing traffic due to its high-bandwidth usage.[65] Compared to Web browsing, e-mail or many other uses of the internet, where data is only transferred in short intervals and relative small quantities, P2P file-sharing often consists of relatively heavy bandwidth usage due to ongoing file transfers and swarm/network coordination packets. In October 2007, Comcast, one of the largest broadband

Internet providers in the United States, started blocking P2P applications such as BitTorrent. Their rationale was that P2P is mostly used to share illegal content, and their infrastructure is not designed for continuous, high-bandwidth traffic. Critics point out that P2P networking has legitimate legal uses, and that this is another way that large providers are trying to control use and content on the Internet, and direct people towards a client–server-based application architecture. The client–server model provides financial barriers-to-entry to small publishers and individuals, and can be less efficient for sharing large files. As a reaction to this bandwidth throttling, several P2P applications started implementing protocol obfuscation, such as the BitTorrent protocol encryption. Techniques for achieving "protocol obfuscation" involves removing otherwise easily identifiable properties of protocols, such as deterministic byte sequences and packet sizes, by making the data look as if it were random.[66] The ISP's solution to the high bandwidth is P2P caching, where an ISP stores the part of files most accessed by P2P clients in order to save access to the Internet.

Current research

Researchers have used computer simulations to aid in understanding and evaluating the complex behaviors of individuals within the network. "Networking research often relies on simulation in order to test and evaluate new ideas. An important requirement of this process is that results must be reproducible so that other researchers can replicate, validate, and extend existing work."[67] If the research cannot be reproduced, then the opportunity for further research is hindered. "Even though new simulators continue to be released, the research community tends towards only a handful of open-source simulators. The demand for features in simulators, as shown by our criteria and survey, is high. Therefore, the community should work together to get these features in open-source software. This would reduce the need for custom simulators, and hence increase repeatability and reputability of experiments."[67]

Besides all the above stated facts, there have been work done on ns-2 open source network simulator. One research issue related to free rider detection and punishment has been explored using ns-2 simulator here.[68]

See also

- Client–queue–client
- Cultural-Historical Activity Theory (CHAT)
- End-to-end principle
- Distributed Data Management Architecture
- List of P2P protocols
- Friend-to-friend
- Peer-to-peer energy trading
- Semantic P2P networks
- Sharing economy
- SponsorChange
- Wireless ad hoc network
- USB dead drop

References

1. Cope, James (2002-04-08). "What's a Peer-to-Peer (P2P) Network?" (https://www.computerworld.com/article/2588287/networking-peer-to-peer-network.html). *Computerworld*. Retrieved 2021-12-21.

2. Rüdiger Schollmeier, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).
3. Bandara, H. M. N. D; A. P. Jayasumana (2012). "Collaborative Applications over Peer-to-Peer Systems – Challenges and Solutions". *Peer-to-Peer Networking and Applications*. **6** (3): 257–276. arXiv:1207.0790 (https://arxiv.org/abs/1207.0790). Bibcode:2012arXiv1207.0790D (https://ui.adsabs.harvard.edu/abs/2012arXiv1207.0790D). doi:10.1007/s12083-012-0157-3 (https://doi.org/10.1007%2Fs12083-012-0157-3). S2CID 14008541 (https://api.semanticscholar.org/CorpusID:14008541).
4. Barkai, David (2001). *Peer-to-peer computing : technologies for sharing and collaborating on the net* (https://archive.org/details/ixp1200programmi00john). Hillsboro, OR: Intel Press. ISBN 978-0970284679. OCLC 49354877 (https://www.worldcat.org/oclc/49354877).
5. Saroiu, Stefan; Gummadi, Krishna P.; Gribble, Steven D. (2003-08-01). "Measuring and analyzing the characteristics of Napster and Gnutella hosts" (https://doi.org/10.1007/s00530-003-0088-1). *Multimedia Systems*. **9** (2): 170–184. doi:10.1007/s00530-003-0088-1 (https://doi.org/10.1007%2Fs00530-003-0088-1). ISSN 1432-1882 (https://www.worldcat.org/issn/1432-1882). S2CID 15963045 (https://api.semanticscholar.org/CorpusID:15963045).
6. Oram, Andrew, ed. (2001). *Peer-to-peer: harnessing the benefits of a disruptive technologies* (https://archive.org/details/peertopeerharnes00oram_0). Sebastopol, California: O'Reilly. ISBN 9780596001100. OCLC 123103147 (https://www.worldcat.org/oclc/123103147).
7. RFC 1, *Host Software*, S. Crocker, IETF Working Group (April 7, 1969)
8. Berners-Lee, Tim (August 1996). "The World Wide Web: Past, Present and Future" (http://www.w3.org/People/Berners-Lee/1996/ppf.html). Retrieved 5 November 2011.
9. Sandhu, R.; Zhang, X. (2005). "Peer-to-peer access control architecture using trusted computing technology" (https://doi.org/10.1145/1063979.1064005). *Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies - SACMAT '05*: 147–158. doi:10.1145/1063979.1064005 (https://doi.org/10.1145%2F1063979.1064005). ISBN 1595930450. S2CID 1478064 (https://api.semanticscholar.org/CorpusID:1478064).
10. Steinmetz, Ralf; Wehrle, Klaus (2005). "2. What Is This "Peer-to-Peer" About?". *Peer-to-Peer Systems and Applications*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. pp. 9–16. doi:10.1007/11530657_2 (https://doi.org/10.1007%2F11530657_2). ISBN 9783540291923.
11. Horton, Mark, and Rick Adams. "Standard for interchange of USENET messages." (1987): 1. https://www.hjp.at/doc/rfc/rfc1036.html
12. Ahson, Syed A.; Ilyas, Mohammad, eds. (2008). *SIP Handbook: Services, Technologies, and Security of Session Initiation Protocol* (https://books.google.com/books?id=CKzPq3-wVdcC&pg=PA204). Taylor & Francis. p. 204. ISBN 9781420066043.
13. Zhu, Ce; et al., eds. (2010). *Streaming Media Architectures: Techniques and Applications: Recent Advances* (https://books.google.com/books?id=Cb4dWYVJ_8AC&pg=PA265). IGI Global. p. 265. ISBN 9781616928339.
14. Kamel, Mina; et al. (2007). "Optimal Topology Design for Overlay Networks" (https://books.google.com/books?id=r4V2G7yPLIAC&pg=PA714). In Akyildiz, Ian F. (ed.). *Networking 2007: Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 6th International IFIP-TC6 Networking Conference, Atlanta, GA, USA, May 14-18, 2007 Proceedings*. Springer. p. 714. ISBN 9783540726050.
15. Filali, Imen; et al. (2011). "A Survey of Structured P2P Systems for RDF Data Storage and Retrieval" (https://books.google.com/books?id=pjQr7BHtbCoC&pg=PA21). In Hameurlain, Abdelkader; et al. (eds.). *Transactions on Large-Scale Data- and Knowledge-Centered Systems III: Special Issue on Data and Knowledge Management in Grid and P2P Systems*. Springer. p. 21. ISBN 9783642230738.

16. Zulhasnine, Mohammed; et al. (2013). "P2P Streaming Over Cellular Networks: Issues, Challenges, and Opportunities" (https://books.google.com/books?id=tr5PGJk-swIC&pg=PA99). In Pathan; et al. (eds.). *Building Next-Generation Converged Networks: Theory and Practice*. CRC Press. p. 99. ISBN 9781466507616.
17. Chervenak, Ann; Bharathi, Shishir (2008). "Peer-to-peer Approaches to Grid Resource Discovery" (https://books.google.com/books?id=adN0pm_BBuYC&pg=PA67). In Danelutto, Marco; et al. (eds.). *Making Grids Work: Proceedings of the CoreGRID Workshop on Programming Models Grid and P2P System Architecture Grid Systems, Tools and Environments 12-13 June 2007, Heraklion, Crete, Greece*. Springer. p. 67. ISBN 9780387784489.
18. Jin, Xing; Chan, S.-H. Gary (2010). "Unstructured Peer-to-Peer Network Architectures". In Shen; et al. (eds.). *Handbook of Peer-to-Peer Networking*. Springer. p. 119. ISBN 978-0-387-09750-3.
19. Lv, Qin; et al. (2002). "Can Heterogeneity Make Gnutella Stable?" (https://books.google.com/books?id=f57AwpUIctcC&pg=PA94). In Druschel, Peter; et al. (eds.). *Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers* (https://archive.org/details/peertopeersystem0000iptp/page/94). Springer. p. 94 (https://archive.org/details/peertopeersystem0000iptp/page/94). ISBN 9783540441793.
20. Shen, Xuemin; Yu, Heather; Buford, John; Akon, Mursalin (2009). *Handbook of Peer-to-Peer Networking* (1st ed.). New York: Springer. p. 118. ISBN 978-0-387-09750-3.
21. Typically approximating O(log N), where N is the number of nodes in the P2P system
22. Other design choices include overlay rings and d-Torus. See for example Bandara, H. M. N. D.; Jayasumana, A. P. (2012). "Collaborative Applications over Peer-to-Peer Systems – Challenges and Solutions". *Peer-to-Peer Networking and Applications*. **6** (3): 257. arXiv:1207.0790 (https://arxiv.org/abs/1207.0790). Bibcode:2012arXiv1207.0790D (https://ui.adsabs.harvard.edu/abs/2012arXiv1207.0790D). doi:10.1007/s12083-012-0157-3 (https://doi.org/10.1007%2Fs12083-012-0157-3). S2CID 14008541 (https://api.semanticscholar.org/CorpusID:14008541).
23. R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: a tutorial," *IEEE Commun. Surv.*, vol. 10, no. 2. and P. Trunfio, "Peer-to-Peer resource discovery in Grids: Models and systems," *Future Generation Computer Systems* archive, vol. 23, no. 7, Aug. 2007.
24. Kelaskar, M.; Matossian, V.; Mehra, P.; Paul, D.; Parashar, M. (2002). *A Study of Discovery Mechanisms for Peer-to-Peer Application* (http://portal.acm.org/citation.cfm?id=873218). pp. 444–. ISBN 9780769515823.
25. Dabek, Frank; Zhao, Ben; Druschel, Peter; Kubiatowicz, John; Stoica, Ion (2003). *Towards a Common API for Structured Peer-to-Peer Overlays*. *Peer-to-Peer Systems II*. Lecture Notes in Computer Science. Vol. 2735. pp. 33–44. CiteSeerX 10.1.1.12.5548 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.5548). doi:10.1007/978-3-540-45172-3_3 (https://doi.org/10.1007%2F978-3-540-45172-3_3). ISBN 978-3-540-40724-9.
26. Moni Naor and Udi Wieder. Novel Architectures for P2P Applications: the Continuous-Discrete Approach (http://www.wisdom.weizmann.ac.il/~naor/PAPERS/dh.pdf). Proc. SPAA, 2003.
27. Gurmeet Singh Manku. Dipsea: A Modular Distributed Hash Table (http://www-db.stanford.edu/~manku/phd/index.html) Archived (https://web.archive.org/web/20040910154927/http://www-db.stanford.edu/~manku/phd/index.html) 2004-09-10 at the Wayback Machine. Ph. D. Thesis (Stanford University), August 2004.
28. Byung-Gon Chun, Ben Y. Zhao, John D. Kubiatowicz (2005-02-24). "Impact of Neighbor Selection on Performance and Resilience of Structured P2P Networks" (https://sites.cs.ucsb.edu/~ravenben/publications/pdf/impact-iptps.pdf) (PDF). Retrieved 2019-08-24.
29. Li, Deng; et al. (2009). Vasilakos, A.V.; et al. (eds.). *An Efficient, Scalable, and Robust P2P Overlay for Autonomic Communication* (https://books.google.com/books?id=c02mTcXW_U4C&pg=PA329). Springer. p. 329. ISBN 978-0-387-09752-7.

30. Bandara, H. M. N. Dilum; Jayasumana, Anura P. (January 2012). "Evaluation of P2P Resource Discovery Architectures Using Real-Life Multi-Attribute Resource and Query Characteristics". *IEEE Consumer Communications and Networking Conf. (CCNC '12)*.
31. Korzun, Dmitry; Gurtov, Andrei (November 2012). *Structured P2P Systems: Fundamentals of Hierarchical Organization, Routing, Scaling, and Security* (https://www.springer.com/gp/book/9781461454823). Springer. ISBN 978-1-4614-5482-3.
32. Ranjan, Rajiv; Harwood, Aaron; Buyya, Rajkumar (1 December 2006). "A Study on Peer-to-Peer Based Discovery of Grid Resource Information" (https://web.archive.org/web/20110514055004/http://www.cs.mu.oz.au/%7Erranjan/pgrid.pdf) (PDF). Archived from the original (http://www.cs.mu.oz.au/%7Erranjan/pgrid.pdf) (PDF) on 14 May 2011. Retrieved 25 August 2008.
33. Ranjan, Rajiv; Chan, Lipo; Harwood, Aaron; Karunasekera, Shanika; Buyya, Rajkumar. "Decentralised Resource Discovery Service for Large Scale Federated Grids" (https://web.archive.org/web/20080910170417/http://gridbus.org/papers/DecentralisedDiscoveryGridFed-eScience2007.pdf) (PDF). Archived from the original (http://gridbus.org/papers/DecentralisedDiscoveryGridFed-eScience2007.pdf) (PDF) on 2008-09-10.
34. Darlagiannis, Vasilios (2005). "Hybrid Peer-to-Peer Systems" (https://books.google.com/books?id=A8CLZ1FB4qoC&pg=PA353). In Steinmetz, Ralf; Wehrle, Klaus (eds.). *Peer-to-Peer Systems and Applications*. Springer. ISBN 9783540291923.
35. Yang, Beverly; Garcia-Molina, Hector (2001). "Comparing Hybrid Peer-to-Peer Systems" (http://infolab.stanford.edu/~byang/pubs/hybridp2p_long.pdf) (PDF). *Very Large Data Bases*. Retrieved 8 October 2013.
36. Padmanabhan, Venkata N.[1] (http://research.microsoft.com/~padmanab/); Sripanidkulchai, Kunwadee [2] (http://www.andrew.cmu.edu/~kunwadee/) (2002). *The Case for Cooperative Networking (PostScript with addendum)* (https://archive.org/details/peertopeersystem0000iptp/page/178). Lecture Notes in Computer Science. Vol. Proceedings of the First International Workshop on Peer-to-Peer Systems. Cambridge, MA: Springer (published March 2002). pp. 178 (https://archive.org/details/peertopeersystem0000iptp/page/178). doi:10.1007/3-540-45748-8_17 (https://doi.org/10.1007%2F3-540-45748-8_17). ISBN 978-3-540-44179-3. {{cite book}}: External link in |first1= and |first2= (help) PDF (Microsoft, with addendum) (http://research.microsoft.com/projects/CoopNet/papers/iptps02-with-addendum.pdf) PDF (Springer, original, fee may be required) (https://doi.org/10.1007%2F3-540-45748-8_17)
37. "CoopNet: Cooperative Networking" (http://research.microsoft.com/projects/CoopNet/). Microsoft Research. Project home page.
38. Vu, Quang H.; et al. (2010). *Peer-to-Peer Computing: Principles and Applications*. Springer. p. 8. ISBN 978-3-642-03513-5.
39. Vu, Quang H.; et al. (2010). *Peer-to-Peer Computing: Principles and Applications*. Springer. pp. 157–159. ISBN 978-3-642-03513-5.
40. Goebel, Jan; et al. (2007). "Measurement and Analysis of Autonomous Spreading Malware in a University Environment" (https://books.google.com/books?id=M0PfEaVa9QIC&pg=PA112). In Hämmerli, Bernhard Markus; Sommer, Robin (eds.). *Detection of Intrusions and Malware, and Vulnerability Assessment: 4th International Conference, DIMVA 2007 Lucerne, Switzerland, July 12-13, 2007 Proceedings*. Springer. p. 112. ISBN 9783540736134.
41. Sorkin, Andrew Ross (4 May 2003). "Software Bullet Is Sought to Kill Musical Piracy" (https://www.nytimes.com/2003/05/04/business/04MUSI.html). *New York Times*. Retrieved 5 November 2011.
42. Singh, Vivek; Gupta, Himani (2012). *Anonymous File Sharing in Peer to Peer System by Random Walks* (Technical report). SRM University. 123456789/9306.
43. Lua, Eng Keong; Crowcroft, Jon; Pias, Marcelo; Sharma, Ravi; Lim, Steven (2005). "A survey and comparison of peer-to-peer overlay network schemes" (https://web.archive.org/web/20120724222234/http://academic.research.microsoft.com/Publication/2633870/a-survey-and-comparison-of-peer-to-peer-overlay-network-schemes). Archived from the original (http://academic.research.microsoft.com/Publication/2633870/a-survey-and-comparison-of-peer-to-peer-overlay-network-schemes) on 2012-07-24.

44. Balakrishnan, Hari; Kaashoek, M. Frans; Karger, David; Morris, Robert; Stoica, Ion (2003). "Looking up data in P2P systems" (http://www.nms.lcs.mit.edu/papers/p43-balakrishnan.pdf) (PDF). *Communications of the ACM*. **46** (2): 43–48. CiteSeerX 10.1.1.5.3597 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.3597). doi:10.1145/606272.606299 (https://doi.org/10.1145%2F606272.606299). S2CID 2731647 (https://api.semanticscholar.org/CorpusID:2731647). Retrieved 8 October 2013.
45. "Art thou a Peer?" (https://web.archive.org/web/20131006022409/http://www.p2pnews.net/2012/06/14/art-thou-a-peer/). *www.p2pnews.net*. 14 June 2012. Archived from the original (http://www.p2pnews.net/2012/06/14/art-thou-a-peer/) on 6 October 2013. Retrieved 10 October 2013.
46. Sharma P., Bhakuni A. & Kaushal R."Performance Analysis of BitTorrent Protocol (http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6488040&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6488040). National Conference on Communications, 2013 doi:10.1109/NCC.2013.6488040 (https://doi.org/10.1109%2FNCC.2013.6488040)
47. Li, Jin (2008). "On peer-to-peer (P2P) content delivery" (http://www.land.ufrj.br/~classes/coppe-redes-2008/biblio/P2P-content-delivery.pdf) (PDF). *Peer-to-Peer Networking and Applications*. **1** (1): 45–63 ≤≥. doi:10.1007/s12083-007-0003-1 (https://doi.org/10.1007%2Fs12083-007-0003-1). S2CID 16438304 (https://api.semanticscholar.org/CorpusID:16438304).
48. Stutzbach, Daniel; et al. (2005). "The scalability of swarming peer-to-peer content delivery" (http://ix.cs.uoregon.edu/~reza/PUB/networking05.pdf) (PDF). In Boutaba, Raouf; et al. (eds.). *NETWORKING 2005 -- Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*. Springer. pp. 15–26. ISBN 978-3-540-25809-4.
49. Gareth Tyson, Andreas Mauthe, Sebastian Kaune, Mu Mu and Thomas Plagemann. Corelli: A Dynamic Replication Service for Supporting Latency-Dependent Content in Community Networks. In Proc. 16th ACM/SPIE Multimedia Computing and Networking Conference (MMCN), San Jose, CA (2009)."Archived copy" (https://web.archive.org/web/20110429181811/http://www.dcs.kcl.ac.uk/staff/tysong/files/MMCN09.pdf) (PDF). Archived from the original (http://www.dcs.kcl.ac.uk/staff/tysong/files/MMCN09.pdf) (PDF) on 2011-04-29. Retrieved 2011-03-12.
50. Glorioso, Andrea; et al. (2010). "The Social Impact of P2P Systems". In Shen; et al. (eds.). *Handbook of Peer-to-Peer Networking*. Springer. p. 48. ISBN 978-0-387-09750-3.
51. John Borland (April 25, 2003). "Judge: File-Swapping Tools are Legal" (https://web.archive.org/web/20120310165410/http://news.cnet.com/Judge-File-swapping-tools-are-legal/2100-1027_3-998363.html). *news.cnet.com*. Archived from the original (http://news.cnet.com/Judge-File-swapping-tools-are-legal/2100-1027_3-998363.html/) on 2012-03-10.
52. Walker, Leslie (2001-11-08). "Uncle Sam Wants Napster!" (https://www.washingtonpost.com/ac2/wp-dyn?pagename=article&node=washtech/techthursday/columns/dotcom&contentId=A59099-2001Nov7). *The Washington Post*. Retrieved 2010-05-22.
53. Hammerksjold Andreas; Engler, Narkis, "Delivery Optimization - a deep dive" (https://channel9.msdn.com/Events/Ignite/Microsoft-Ignite-Orlando-2017/BRK2048), *Channel 9*, 11 October 2017, Retrieved on 4 February 2019.
54. Krishnan, R., Smith, M. D., Tang, Z., & Telang, R. (2004, January). The impact of free-riding on peer-to-peer networks. In System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on (pp. 10-pp). IEEE.
55. Feldman, M., Lai, K., Stoica, I., & Chuang, J. (2004, May). Robust incentive techniques for peer-to-peer networks. In Proceedings of the 5th ACM conference on Electronic commerce (pp. 102-111). ACM.
56. Vu, Quang H.; et al. (2010). *Peer-to-Peer Computing: Principles and Applications*. Springer. p. 172. ISBN 978-3-642-03513-5.
57. P. Antoniadis and B. Le Grand, "Incentives for resource sharing in self-organized communities: From economics to social psychology," Digital Information Management (ICDIM '07), 2007

58. Vu, Quang H.; et al. (2010). *Peer-to-Peer Computing: Principles and Applications*. Springer. pp. 179–181. ISBN 978-3-642-03513-5.
59. "No country is free from child sexual abuse, exploitation, UN's top rights forum hears" (https://news.un.org/en/story/2020/03/1058501). *UN News*. March 3, 2020.
60. Majoras, D. B. (2005). Peer-to-peer file-sharing technology consumer protection and competition issues. Federal Trade Commission, Retrieved from http://www.ftc.gov/reports/p2p05/050623p2prpt.pdf
61. The Government of the Hong Kong Special Administrative Region, (2008). Peer-to-peer network. Retrieved from website: http://www.infosec.gov.hk/english/technical/files/peer.pdf
62. Sanders, Linley (2017-09-22). "Illegal downloads may not actually harm sales, but the European Union doesn't want you to know that" (http://www.newsweek.com/secret-piracy-study-european-union-669436). *Newsweek*. Retrieved 2018-03-29.
63. Polgar, David Ryan (October 15, 2017). "Does Video Game Piracy Actually Result in More Sales?" (http://bigthink.com/david-ryan-polgar/video-game-piracy-may-actually-result-in-more-sales). *Big Think*. Retrieved 2018-03-29.
64. Orland, Kyle (September 26, 2017). "EU study finds piracy doesn't hurt game sales, may actually help" (https://arstechnica.com/gaming/2017/09/eu-study-finds-piracy-doesnt-hurt-game-sales-may-actually-help/). *Ars Technica*. Retrieved 2018-03-29.
65. Janko Roettgers, 5 Ways to Test Whether your ISP throttles P2P, http://newteevee.com/2008/04/02/5-ways-to-test-if-your-isp-throttles-p2p/
66. Hjelmvik, Erik; John, Wolfgang (2010-07-27). "Breaking and Improving Protocol Obfuscation" (http://www.iis.se/docs/hjelmvik_breaking.pdf) (PDF). *Technical Report*. ISSN 1652-926X (https://www.worldcat.org/issn/1652-926X).
67. Basu, A., Fleming, S., Stanier, J., Naicken, S., Wakeman, I., & Gurbani, V. K. (2013). The state of peer-to-peer network simulators. ACM Computing Surveys, 45(4), 46.
68. A Bhakuni, P Sharma, R Kaushal "Free-rider detection and punishment in BitTorrent based P2P networks" (http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6779311&tag=1), International Advanced Computing Conference, 2014. doi:10.1109/IAdCC.2014.6779311 (https://doi.org/10.1109%2FIAdCC.2014.6779311)

External links

- Ghosh Debjani, Rajan Payas, Pandey Mayank P2P-VoD Streaming: Design Issues & User Experience Challenges (https://link.springer.com/chapter/10.1007%2F978-3-319-07350-7_19) Springer Proceedings, June 2014
- Glossary (https://web.archive.org/web/20090922214835/http://www.p2pna.com/glossary.html) of P2P terminology
- Foundation of Peer-to-Peer Computing (https://web.archive.org/web/20090217141525/http://www.sciencedirect.com/science/issue/5624-2008-999689997-678759), Special Issue, Elsevier Journal of Computer Communication, (Ed) Javed I. Khan and Adam Wierzbicki, Volume 31, Issue 2, February 2008
- Anderson, Ross J. "The eternity service" (http://www.cl.cam.ac.uk/users/rja14/eternity/eternity.html). *Pragocrypt*. **1996**.
- Marling Engle & J. I. Khan. Vulnerabilities of P2P systems and a critical look at their solutions (http://www.medianet.kent.edu/techreports/TR2006-11-01-p2pvuln-EK.pdf), May 2006
- Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies (http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html). ACM Computing Surveys, 36(4):335–371, December 2004.
- Biddle, Peter, Paul England, Marcus Peinado, and Bryan Willman, The Darknet and the Future of Content Distribution (http://crypto.stanford.edu/DRM2002/darknet5.doc). In *2002 ACM Workshop on Digital Rights Management*, November 2002.

- John F. Buford, Heather Yu, Eng Keong Lua P2P Networking and Applications (https://web.archive.org/web/20090211135759/http://p2pna.com/). ISBN 0123742145, Morgan Kaufmann, December 2008
- Djamal-Eddine Meddour, Mubashar Mushtaq, and Toufik Ahmed, "Open Issues in P2P Multimedia Streaming (http://multicomm.polito.it/proc_multicomm06_8.pdf)", in the proceedings of the 1st Multimedia Communications Workshop MULTICOMM 2006 held in conjunction with IEEE ICC 2006 pp 43–48, June 2006, Istanbul, Turkey.
- Detlef Schoder and Kai Fischbach, "Core Concepts in Peer-to-Peer (P2P) Networking (http://www.econbiz.de/archiv1/2008/42151_concepts_peer-to-peer_networking.pdf)". In: Subramanian, R.; Goodman, B. (eds.): *P2P Computing: The Evolution of a Disruptive Technology*, Idea Group Inc, Hershey. 2005
- Ramesh Subramanian and Brian Goodman (eds), *Peer-to-Peer Computing: Evolution of a Disruptive Technology (https://web.archive.org/web/20070928224509/http://www.igi-pub.com/books/details.asp?ID=4635)*, ISBN 1-59140-429-0, Idea Group Inc., Hershey, PA, United States, 2005.
- Shuman Ghosemajumder. Advanced Peer-Based Technology Business Models (http://dspace.mit.edu/handle/1721.1/8438). MIT Sloan School of Management, 2002.
- Silverthorne, Sean. *Music Downloads: Pirates- or Customers? (http://hbswk.hbs.edu/item.jhtml?id=4206&t=innovation)*. Harvard Business School Working Knowledge, 2004.
- Glasnost (https://broadband.mpi-sws.org/transparency/bttest.php) test P2P traffic shaping (Max Planck Institute for Software Systems)

Consensus (computer science)

A fundamental problem in distributed computing and multi-agent systems is to achieve overall system reliability in the presence of a number of faulty processes. This often requires coordinating processes to reach **consensus**, or agree on some data value that is needed during computation. Example applications of consensus include agreeing on what transactions to commit to a database in which order, state machine replication, and atomic broadcasts. Real-world applications often requiring consensus include cloud computing, clock synchronization, PageRank, opinion formation, smart power grids, state estimation, control of UAVs (and multiple robots/agents in general), load balancing, blockchain, and others.

Contents

Problem description

The consensus problem requires agreement among a number of processes (or agents) for a single data value. Some of the processes (agents) may fail or be unreliable in other ways, so consensus protocols must be fault tolerant or resilient. The processes must somehow put forth their candidate values, communicate with one another, and agree on a single consensus value.

The consensus problem is a fundamental problem in control of multi-agent systems. One approach to generating consensus is for all processes (agents) to agree on a majority value. In this context, a majority requires at least one more than half of available votes (where each process is given a vote).

However, one or more faulty processes may skew the resultant outcome such that consensus may not be reached or reached incorrectly.

Protocols that solve consensus problems are designed to deal with limited numbers of faulty processes. These protocols must satisfy a number of requirements to be useful. For instance, a trivial protocol could have all processes output binary value 1. This is not useful and thus the requirement is modified such that the output must somehow depend on the input. That is, the output value of a consensus protocol must be the input value of some process. Another requirement is that a process may decide upon an output value only once and this decision is irrevocable. A process is called correct in an execution if it does not experience a failure. A consensus protocol tolerating halting failures must satisfy the following properties.[1]

Termination
: Eventually, every correct process decides some value.

Integrity
: If all the correct processes proposed the same value v, then any correct process must decide v.

Agreement
: Every correct process must agree on the same value.

Variations on the definition of *integrity* may be appropriate, according to the application. For example, a weaker type of integrity would be for the decision value to equal a value that some correct process proposed – not necessarily all of them.[1] There is also a condition known as **validity** in the literature which refers to the property that a message sent by a process must be delivered.[1]

A protocol that can correctly guarantee consensus amongst n processes of which at most t fail is said to be *t-resilient*.

In evaluating the performance of consensus protocols two factors of interest are *running time* and *message complexity*. Running time is given in Big O notation in the number of rounds of message exchange as a function of some input parameters (typically the number of processes and/or the size of the input domain). Message complexity refers to the amount of message traffic that is generated by the protocol. Other factors may include memory usage and the size of messages.

Models of computation

Varying models of computation may define a "consensus problem". Some models may deal with fully connected graphs, while others may deal with rings and trees. In some models message authentication is allowed, whereas in others processes are completely anonymous. Shared memory models in which processes communicate by accessing objects in shared memory are also an important area of research.

Communication channels with direct or transferable authentication

In most models of communication protocol participants communicate through *authenticated channels*. This means that messages are not anonymous, and receivers know the source of every message they receive. Some models assume a stronger, *transferable* form of authentication, where each *message* is signed by the sender, so that a receiver knows not just the immediate source of every message, but the participant that initially created the message. This stronger type of authentication is achieved by digital signatures, and when this stronger form of authentication is available, protocols can tolerate a larger number of faults.[2]

The two different authentication models are often called *oral communication* and *written communication* models. In an oral communication model, the immediate source of information is known, whereas in stronger, written communication models, every step along the receiver learns not just the immediate source of the message, but the communication history of the message.[3]

Inputs and outputs of consensus

In the most traditional **single-value** consensus protocols such as Paxos, cooperating nodes agree on a single value such as an integer, which may be of variable size so as to encode useful metadata such as a transaction committed to a database.

A special case of the single-value consensus problem, called **binary consensus**, restricts the input, and hence the output domain, to a single binary digit {0,1}. While not highly useful by themselves, binary consensus protocols are often useful as building blocks in more general consensus protocols, especially for asynchronous consensus.

In **multi-valued** consensus protocols such as Multi-Paxos and Raft, the goal is to agree on not just a single value but a series of values over time, forming a progressively-growing history. While multi-valued consensus may be achieved naively by running multiple iterations of a single-valued consensus protocol in succession, many optimizations and other considerations such as reconfiguration support can make multi-valued consensus protocols more efficient in practice.

Crash and Byzantine failures

There are two types of failures a process may undergo, a crash failure or a Byzantine failure. A *crash failure* occurs when a process abruptly stops and does not resume. *Byzantine failure*s are failures in which absolutely no conditions are imposed. For example, they may occur as a result of the malicious actions of an adversary. A process that experiences a Byzantine failure may send contradictory or conflicting data to other processes, or it may sleep and then resume activity after a lengthy delay. Of the two types of failures, Byzantine failures are far more disruptive.

Thus, a consensus protocol tolerating Byzantine failures must be resilient to every possible error that can occur.

A stronger version of consensus tolerating Byzantine failures is given by strengthening the Integrity constraint:

Integrity
: If a correct process decides v, then v must have been proposed by some correct process.

Asynchronous and synchronous systems

The consensus problem may be considered in the case of asynchronous or synchronous systems. While real world communications are often inherently asynchronous, it is more practical and often easier to model synchronous systems,[4] given that asynchronous systems naturally involve more issues than synchronous ones.

In synchronous systems, it is assumed that all communications proceed in *rounds*. In one round, a process may send all the messages it requires, while receiving all messages from other processes. In this manner, no message from one round may influence any messages sent within the same round.

The FLP impossibility result for asynchronous deterministic consensus

In a fully asynchronous message-passing distributed system, in which at least one process may have a *crash failure*, it has been proven in the famous **FLP impossibility result** that a deterministic algorithm for achieving consensus is impossible.[5] This impossibility result derives from worst-case scheduling scenarios, which are unlikely to occur in practice except in adversarial situations such as an intelligent denial-of-service attacker in the network. In most normal situations, process scheduling has a degree of natural randomness.[4]

In an asynchronous model, some forms of failures can be handled by a synchronous consensus protocol. For instance, the loss of a communication link may be modeled as a process which has suffered a Byzantine failure.

Randomized consensus algorithms can circumvent the FLP impossibility result by achieving both safety and liveness with overwhelming probability, even under worst-case scheduling scenarios such as an intelligent denial-of-service attacker in the network.[6]

Permissioned versus permissionless consensus

Consensus algorithms traditionally assume that the set of participating nodes is fixed and given at the outset: that is, that some prior (manual or automatic) configuration process has **permissioned** a particular known group of participants who can authenticate each other as members of the group. In the absence of such a well-defined, closed group with authenticated members, a Sybil attack against an open consensus group can defeat even a Byzantine consensus algorithm, simply by creating enough virtual participants to overwhelm the fault tolerance threshold.

A **permissionless** consensus protocol, in contrast, allows anyone in the network to join dynamically and participate without prior permission, but instead imposes a different form of artificial cost or barrier to entry to mitigate the Sybil attack threat. Bitcoin introduced the first permissionless consensus protocol using proof of work and a difficulty adjustment function, in which participants compete to solve cryptographic hash puzzles, and probabilistically earn the right to commit blocks and earn associated rewards in proportion to their invested computational effort. Motivated in part by the high energy cost of this approach, subsequent permissionless consensus protocols have proposed or adopted other alternative participation rules for Sybil attack protection, such as proof of stake, proof of space, and proof of authority.

Equivalency of agreement problems

Three agreement problems of interest are as follows.

Terminating Reliable Broadcast

A collection of n processes, numbered from 0 to $n-1$, communicate by sending messages to one another. Process 0 must transmit a value v to all processes such that:

1. if process 0 is correct, then every correct process receives v
2. for any two correct processes, each process receives the same value.

It is also known as The General's Problem.

Consensus

Formal requirements for a consensus protocol may include:

- *Agreement*: All correct processes must agree on the same value.
- *Weak validity*: For each correct process, its output must be the input of some correct process.
- *Strong validity*: If all correct processes receive the same input value, then they must all output that value.
- *Termination*: All processes must eventually decide on an output value

Weak Interactive Consistency

For n processes in a partially synchronous system (the system alternates between good and bad periods of synchrony), each process chooses a private value. The processes communicate with each other by rounds to determine a public value and generate a consensus vector with the following requirements:[7]

1. if a correct process sends v, then all correct processes receive either v or nothing (integrity property)
2. all messages sent in a round by a correct process are received in the same round by all correct processes (consistency property).

It can be shown that variations of these problems are equivalent in that the solution for a problem in one type of model may be the solution for another problem in another type of model. For example, a solution to the Weak Byzantine General problem in a synchronous authenticated message passing model leads to a solution for Weak Interactive Consistency.[8] An interactive consistency algorithm can solve the consensus problem by having each process choose the majority value in its consensus vector as its consensus value.[9]

Solvability results for some agreement problems

There is a t-resilient anonymous synchronous protocol which solves the Byzantine Generals problem,[10][11] if $\frac{t}{n} < \frac{1}{3}$ and the Weak Byzantine Generals case[8] where t is the number of failures and n is the number of processes.

For systems with n processors, of which f are Byzantine, it has been shown that there exists no algorithm that solves the consensus problem for $n \leq 3f$ in the *oral-messages model*.[12] The proof is constructed by first showing the impossibility for the three-node case $n = 3$ and using this result to argue about partitions of processors. In the *written-messages model* there are protocols that can tolerate $n = f + 1$.[2]

In a fully asynchronous system there is no consensus solution that can tolerate one or more crash failures even when only requiring the non triviality property.[5] This result is sometimes called the FLP impossibility proof named after the authors Michael J. Fischer, Nancy Lynch, and Mike Paterson who were awarded a Dijkstra Prize for this significant work. The FLP result has been mechanically verified to hold even under fairness assumptions.[13] However, FLP does not state that consensus can never be reached: merely that under the model's assumptions, no algorithm can always reach consensus in bounded time. In practice it is highly unlikely to occur.

Some consensus protocols

The Paxos consensus algorithm by Leslie Lamport, and variants of it such as Raft, are used pervasively in widely deployed distributed and cloud computing systems. These algorithms are typically synchronous, dependent on an elected leader to make progress, and tolerate only crashes and not Byzantine failures.

An example of a polynomial time binary consensus protocol that tolerates Byzantine failures is the Phase King algorithm[14] by Garay and Berman. The algorithm solves consensus in a synchronous message passing model with n processes and up to f failures, provided $n > 4f$. In the phase king algorithm, there are $f + 1$ phases, with 2 rounds per phase. Each process keeps track of its preferred output (initially equal to the process's own input value). In the first round of each phase each process broadcasts its own preferred value to all other processes. It then receives the values from all processes and determines which value is the majority value and its count. In the second round of the phase, the process whose id matches the current phase number is designated the king of the phase. The king broadcasts the majority value it observed in the first round and serves as a tie breaker. Each process then updates its preferred value as follows. If the count of the majority value the process observed in the first round is greater than $n/2 + f$, the process changes its preference to that majority value; otherwise it uses the phase king's value. At the end of $f + 1$ phases the processes output their preferred values.

Google has implemented a distributed lock service library called Chubby.[15] Chubby maintains lock information in small files which are stored in a replicated database to achieve high availability in the face of failures. The database is implemented on top of a fault-tolerant log layer which is based on the Paxos consensus algorithm. In this scheme, Chubby clients communicate with the Paxos *master* in order to access/update the replicated log; i.e., read/write to the files.[16]

Many peer-to-peer online Real-time strategy games use a modified Lockstep protocol as a consensus protocol in order to manage game state between players in a game. Each game action results in a game state delta broadcast to all other players in the game along with a hash of the total game state. Each player validates the change by applying the delta to their own game state and comparing the game state hashes. If the hashes do not agree then a vote is cast, and those players whose game state is in the minority are disconnected and removed from the game (known as a desync.)

Another well-known approach is called MSR-type algorithms which have been used widely from computer science to control theory.[17][18][19]

Source	Synchrony	Authentication	Threshold	Rounds	Notes
Pease-Shostak-Lamport [10]	Synchronous	Oral	$n > 3f$	$f + 1$	total communication $O(n^f)$
Pease-Shostak-Lamport [10]	Synchronous	Written	$n > f + 1$	$f + 1$	total communication $O(n^f)$
Ben-Or [20]	Asynchronous	Oral	$n > 5f$	$O(2^n)$ (expected)	expected $O(1)$ rounds when $f < \sqrt{n}$
Dolev et al.[21]	Synchronous	Oral	$n > 3f$	$2f + 3$	total communication $O(f^3 \log f)$
Dolev-Strong [2]	Synchronous	Written	$n > f + 1$	$f + 1$	total communication $O(n^2)$
Dolev-Strong [2]	Synchronous	Written	$n > f + 1$	$f + 2$	total communication $O(nf)$
Feldman-Micali [22]	Synchronous	Oral	$n > 3f$	$O(1)$ (expected)	
Katz-Koo [23]	Synchronous	Written	$n > 2f$	$O(1)$ (expected)	Requires PKI
PBFT [24]	Asynchronous (safety)-- Synchronous (liveness)	Oral	$n > 3f$		
HoneyBadger [25]	Asynchronous	Oral	$n > 3f$	$O(\log n)$ (expected)	per tx communication $O(n)$ - requires public-key encryption
Abraham et al.[26]	Synchronous	Written	$n > 2f$	8	
Byzantine Agreement Made Trivial [27][28]	Synchronous	Signatures	$n > 3f$	9 (expected)	Requires digital signatures

Permissionless consensus protocols

Bitcoin uses proof of work, a difficulty adjustment function and a reorganization function to achieve permissionless consensus in its open peer-to-peer network. To extend Bitcoin's blockchain or distributed ledger, *miners* attempt to solve a cryptographic puzzle, where probability of finding a solution is proportional to the computational effort expended in hashes per second. The node that first solves such a puzzle has their proposed version of the next block of transactions added to the ledger and eventually accepted by all other nodes. As any node in the network can attempt to solve the proof-of-work problem, a Sybil attack is infeasible in principle unless the attacker has over 50% of the computational resources of the network.

Other cryptocurrencies (i.e. NEO, STRATIS, ...) use proof of stake, in which nodes compete to append blocks and earn associated rewards in proportion to *stake*, or existing cryptocurrency allocated and locked or *staked* for some time period. One advantage of a 'proof of stake' over a

'proof of work' system, is the high energy consumption demanded by the latter. As an example, Bitcoin mining (2018) is estimated to consume non-renewable energy sources at an amount similar to the entire nations of Czech Republic or Jordan.[29]

Some cryptocurrencies, such as Ripple, use a system of validating nodes to validate the ledger. This system used by Ripple, called Ripple Protocol Consensus Algorithm (RPCA), works in rounds: Step 1: every server compiles a list of valid candidate transactions; Step 2: each server amalgamates all candidates coming from its Unique Nodes List (UNL) and votes on their veracity; Step 3: transactions passing the minimum threshold are passed to the next round; Step 4: the final round requires 80% agreement[30]

Other participation rules used in permissionless consensus protocols to impose barriers to entry and resist sybil attacks include proof of authority, proof of space, proof of burn, or proof of elapsed time. These alternatives are again largely motivated by the high amount of computational energy consumed by the proof of work.[31] Proof of space is used by cryptocoins such as Burstcoin.

Contrasting with the above permissionless participation rules, all of which reward participants in proportion to amount of investment in some action or resource, proof of personhood protocols aim to give each real human participant exactly one unit of voting power in permissionless consensus, regardless of economic investment.[32][33] Proposed approaches to achieving one-per-person distribution of consensus power for proof of personhood include physical pseudonym parties,[34] social networks,[35] pseudonymized government-issued identities,[36] and biometrics.[37]

Consensus number

To solve the consensus problem in a shared-memory system, concurrent objects must be introduced. A concurrent object, or shared object, is a data structure which helps concurrent processes communicate to reach an agreement. Traditional implementations using critical sections face the risk of crashing if some process dies inside the critical section or sleeps for an intolerably long time. Researchers defined wait-freedom as the guarantee that the algorithm completes in a finite number of steps.

The **consensus number** of a concurrent object is defined to be the maximum number of processes in the system which can reach consensus by the given object in a wait-free implementation.[38] Objects with a consensus number of n can implement any object with a consensus number of n or lower, but cannot implement any objects with a higher consensus number. The consensus numbers form what is called Herlihy's hierarchy of synchronization objects.[39]

Consensus number	Objects
1	atomic read/write registers, mutex
2	test-and-set, swap, fetch-and-add, wait-free queue or stack
...	...
$2n-2$	n-register assignment
...	...
∞	compare-and-swap, load-link/store-conditional,[40] memory-to-memory move and swap, queue with peek operation, fetch&cons, sticky byte

According to the hierarchy, read/write registers cannot solve consensus even in a 2-process system. Data structures like stacks and queues can only solve consensus between two processes. However, some concurrent objects are universal (notated in the table with ∞), which means they can solve

consensus among any number of processes and they can simulate any other objects through an operation sequence.[38]

See also

- Uniform consensus
- Quantum Byzantine agreement
- Byzantine fault tolerance

References

1. Coulouris, George; Jean Dollimore; Tim Kindberg (2001), *Distributed Systems: Concepts and Design (3rd Edition)*, Addison-Wesley, p. 452, ISBN 978-0201-61918-8
2. Dolev, D.; Strong, H.R. (1983). "Authenticated algorithms for Byzantine agreement". *SIAM Journal on Computing*. **12** (4): 656–666. doi:10.1137/0212045 (https://doi.org/10.1137%2F0212045).
3. Gong, Li; Lincoln, Patrick; Rushby, John (1995). "Byzantine Agreement with authentication" (http://www.csl.sri.com/papers/dcca95/). *Dependable Computing for Critical Applications*. **10**.
4. Aguilera, M. K. (2010). "Stumbling over Consensus Research: Misunderstandings and Issues". *Replication*. Lecture Notes in Computer Science. Vol. 5959. pp. 59–72. doi:10.1007/978-3-642-11294-2_4 (https://doi.org/10.1007%2F978-3-642-11294-2_4). ISBN 978-3-642-11293-5.
5. Fischer, M. J.; Lynch, N. A.; Paterson, M. S. (1985). "Impossibility of distributed consensus with one faulty process" (https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf) (PDF). *Journal of the ACM*. **32** (2): 374–382. doi:10.1145/3149.214121 (https://doi.org/10.1145%2F3149.214121). S2CID 207660233 (https://api.semanticscholar.org/CorpusID:207660233).
6. Aspnes, James (May 1993). "Time- and Space-Efficient Randomized Consensus" (https://www.sciencedirect.com/science/article/abs/pii/S0196677483710229). *Journal of Algorithms*. **14** (3): 414–431. doi:10.1006/jagm.1993.1022 (https://doi.org/10.1006%2Fjagm.1993.1022).
7. Milosevic, Zarko; Martin Hutle; Andre Schiper (2009). *Unifying Byzantine Consensus Algorithms with Weak Interactive Consistency* (https://archive.org/details/principlesofdist0000opod/page/300). *Principles of Distributed Systems, Lecture Notes in Computer Science*. Lecture Notes in Computer Science. Vol. 5293. pp. 300–314 (https://archive.org/details/principlesofdist0000opod/page/300). CiteSeerX 10.1.1.180.4229 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.180.4229). doi:10.1007/978-3-642-10877-8_24 (https://doi.org/10.1007%2F978-3-642-10877-8_24). ISBN 978-3-642-10876-1.
8. Lamport, L. (1983). "The Weak Byzantine Generals Problem". *Journal of the ACM*. **30** (3): 668. doi:10.1145/2402.322398 (https://doi.org/10.1145%2F2402.322398). S2CID 1574706 (https://api.semanticscholar.org/CorpusID:1574706).
9. Fischer, Michael J. "The Consensus Problem in Unreliable Distributed Systems (A Brief Survey)" (http://zoo.cs.yale.edu/classes/cs426/2012/bib/fischer83consensus.pdf) (PDF). Retrieved 21 April 2014.
10. Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" (http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf) (PDF). *ACM Transactions on Programming Languages and Systems*. **4** (3): 382–401. CiteSeerX 10.1.1.64.2312 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.64.2312). doi:10.1145/357172.357176 (https://doi.org/10.1145%2F357172.357176). S2CID 55899582 (https://api.semanticscholar.org/CorpusID:55899582).

11. Lamport, Leslie; Marshall Pease; Robert Shostak (April 1980). "Reaching Agreement in the Presence of Faults" (http://research.microsoft.com/users/lamport/pubs/reaching.pdf) (PDF). *Journal of the ACM*. **27** (2): 228–234. CiteSeerX 10.1.1.68.4044 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.4044). doi:10.1145/322186.322188 (https://doi.org/10.1145%2F322186.322188). S2CID 6429068 (https://api.semanticscholar.org/CorpusID:6429068). Retrieved 2007-07-25.
12. Attiya, Hagit (2004). *Distributed Computing 2nd Ed*. Wiley. pp. 101–103. ISBN 978-0-471-45324-6.
13. Bisping, Benjamin; et al. (2016), Blanchette, Jasmin Christian; Merz, Stephan (eds.), *Mechanical Verification of a Constructive Proof for FLP*, Lecture Notes in Computer Science, vol. 9807, Springer International Publishing, doi:10.1007/978-3-319-43144-4_7 (https://doi.org/10.1007%2F978-3-319-43144-4_7), ISBN 978-3-319-43144-4
14. Berman, Piotr; Juan A. Garay (1993). "Cloture Votes: n/4-resilient Distributed Consensus in t + 1 rounds". *Theory of Computing Systems*. 2. **26**: 3–19. doi:10.1007/BF01187072 (https://doi.org/10.1007%2FBF01187072). S2CID 6102847 (https://api.semanticscholar.org/CorpusID:6102847).
15. Burrows, M. (2006). *The Chubby lock service for loosely-coupled distributed systems* (http://research.google.com/archive/chubby-osdi06.pdf) (PDF). Proceedings of the 7th Symposium on Operating Systems Design and Implementation. USENIX Association Berkeley, CA, USA. pp. 335–350.
16. C., Tushar; Griesemer, R; Redstone J. (2007). *Paxos Made Live – An Engineering Perspective* (https://web.archive.org/web/20141212001817/http://www.cs.cmu.edu/~pavlo/courses/fall2013/static/papers/p398-chandra.pdf) (PDF). Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing. Portland, Oregon, USA: ACM Press New York, NY, USA. pp. 398–407. doi:10.1145/1281100.1281103 (https://doi.org/10.1145%2F1281100.1281103). Archived from the original (http://delivery.acm.org/10.1145/1290000/1281103/p398-chandra.pdf?key1=1281103&key2=4382532021&coll=GUIDE&dl=GUIDE&CFID=15324100&CFTOKEN=95510390) (PDF) on 2014-12-12. Retrieved 2008-02-06.
17. LeBlanc, Heath J. (April 2013). "Resilient Asymptotic Consensus in Robust Networks". *IEEE Journal on Selected Areas in Communications*. **31** (4): 766–781. CiteSeerX 10.1.1.310.5354 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.310.5354). doi:10.1109/JSAC.2013.130413 (https://doi.org/10.1109%2FJSAC.2013.130413). S2CID 11287513 (https://api.semanticscholar.org/CorpusID:11287513).
18. Dibaji, S. M. (May 2015). "Consensus of second-order multi-agent systems in the presence of locally bounded faults". *Systems & Control Letters*. **79**: 23–29. doi:10.1016/j.sysconle.2015.02.005 (https://doi.org/10.1016%2Fj.sysconle.2015.02.005).
19. Dibaji, S. M. (July 2017). "Resilient consensus of second-order agent networks: Asynchronous update rules with delays". *Automatica*. **81**: 123–132. arXiv:1701.03430 (https://arxiv.org/abs/1701.03430). Bibcode:2017arXiv170103430M (https://ui.adsabs.harvard.edu/abs/2017arXiv170103430M). doi:10.1016/j.automatica.2017.03.008 (https://doi.org/10.1016%2Fj.automatica.2017.03.008). S2CID 7467466 (https://api.semanticscholar.org/CorpusID:7467466).
20. Ben-Or, Michael (1983). "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols". *Proceedings of the second annual ACM symposium on Principles of distributed computing*. pp. 27–30. doi:10.1145/800221.806707 (https://doi.org/10.1145%2F800221.806707). S2CID 38215511 (https://api.semanticscholar.org/CorpusID:38215511).
21. Dolev, Danny; Fisher, Michael J.; Fowler, Rob; Lynch, Nancy; Strong, H. Raymond (1982). "An Efficient Algorithm for Byzantine Agreement without Authentication". *Information and Control*. **52** (3): 257–274. doi:10.1016/S0019-9958(82)90776-8 (https://doi.org/10.1016%2FS0019-9958%2882%2990776-8).
22. Feldman, Pesech; Micali, Sylvio (1997). *An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM Journal on Computing* (Technical report). doi:10.1137/S0097539790187084 (https://doi.org/10.1137%2FS0097539790187084).

23. Katz, Jonathan; Koo, Chiu-Yuen (2006). *On Expected Constant-Round Protocols for Byzantine Agreement*. CRYPTO. doi:10.1007/11818175_27 (https://doi.org/10.1007%2F11818175_27).
24. Castro, Miguel; Liskov, Barbara (1999). *Practical Byzantine Fault Tolerance* (http://pmg.csail.mit.edu/papers/osdi99.pdf) (PDF). OSDI.
25. Miller, Andrew; Xia, Yu; Croman, Kyle; Shi, Elaine; Song, Dawn (2016). *The honey badger of BFT protocols* (https://eprint.iacr.org/2016/199). CCS. doi:10.1145/2976749.2978399 (https://doi.org/10.1145%2F2976749.2978399).
26. Abraham, Ittai; Devadas, Srinivas; Dolev, Danny; Nayak, Kartik; Ren, Ling (2017). *Efficient Synchronous Byzantine Consensus* (https://eprint.iacr.org/2017/307) (Technical report).
27. Micali, Sylvio (2018). "Byzantine agreement made trivial" (https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Distributed%20Computation/BYZANTYNE%20AGREEMENT%20MADE%20TRIVIAL.pdf) (PDF).
28. Chen, Jing; Micali, Silvio (2016). "ALGORAND". arXiv:1607.01341v9 (https://arxiv.org/abs/1607.01341v9) [cs.CR (https://arxiv.org/archive/cs.CR)].
29. Irfan, Umair (June 18, 2019). "Bitcoin is an energy hog. Where is all that electricity coming from?" (https://www.vox.com/2019/6/18/18642645/bitcoin-energy-price-renewable-china). *Vox*.
30. Schwartz D, YoungsN, Britto A. 2014 The Ripple protocol consensus algorithm
31. What are the alternative strategies for proof of work? (https://www.blockchain-council.org/blockchain/what-are-the-alternative-strategies-for-proof-of-work/)
32. Maria Borge, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Bryan Ford (29 April 2017). *Proof-of-Personhood: Redemocratizing Permissionless Cryptocurrencies* (https://ieeexplore.ieee.org/document/7966966). IEEE Security & Privacy on the Blockchain (IEEE S&B) (https://prosecco.gforge.inria.fr/ieee-blockchain2016/). doi:10.1109/EuroSPW.2017.46 (https://doi.org/10.1109%2FEuroSPW.2017.46).
33. Divya Siddarth, Sergey Ivliev, Santiago Siri, Paula Berman (13 Oct 2020). "Who Watches the Watchmen? A Review of Subjective Approaches for Sybil-resistance in Proof of Personhood Protocols". arXiv:2008.05300 (https://arxiv.org/abs/2008.05300) [cs.CR (https://arxiv.org/archive/cs.CR)].
34. Ford, Bryan; Strauss, Jacob (1 April 2008). *An Offline Foundation for Online Accountable Pseudonyms* (https://dl.acm.org/doi/10.1145/1435497.1435503). 1st Workshop on Social Network Systems - SocialNets '08 (https://dl.acm.org/doi/proceedings/10.1145/1435497). pp. 31–6. doi:10.1145/1435497.1435503 (https://doi.org/10.1145%2F1435497.1435503). ISBN 978-1-60558-124-8.
35. Gal Shahaf, Ehud Shapiro, Nimrod Talmon (October 2020). *Genuine Personal Identifiers and Mutual Sureties for Sybil-Resilient Community Growth* (https://link.springer.com/chapter/10.1007/978-3-030-60975-7_24). International Conference on Social Informatics (https://kdd.isti.cnr.it/socinfo2020/index.html). doi:10.1007/978-3-030-60975-7_24 (https://doi.org/10.1007%2F978-3-030-60975-7_24).
36. Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, Andrew Miller (28 Sep 2020). "CanDID: Can-Do Decentralized Identity with Legacy Compatibility, Sybil-Resistance, and Accountability" (https://eprint.iacr.org/2020/934.pdf) (PDF).
37. Mohammad-Javad Hajialikhani, Mohammad-Mahdi Jahanara (20 June 2018). "UniqueID: Decentralized Proof-of-Unique-Human". arXiv:1806.07583 (https://arxiv.org/abs/1806.07583) [cs.CR (https://arxiv.org/archive/cs.CR)].
38. Herlihy, Maurice. "Wait-Free Synchronization" (http://www.cs.brown.edu/~mph/Herlihy91/p124-herlihy.pdf) (PDF). Retrieved 19 December 2011.

39. Imbs, Damien; Raynal, Michel (25 July 2010). "The multiplicative power of consensus numbers" (https://hal.inria.fr/inria-00454399/file/PI-1949.pdf) (PDF). *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. Association for Computing Machinery: 26–35. doi:10.1145/1835698.1835705 (https://doi.org/10.1145%2F1835698.1835705). ISBN 9781605588889. S2CID 3179361 (https://api.semanticscholar.org/CorpusID:3179361). Retrieved 22 April 2021.
40. Fich, Faith; Hendler, Danny; Shavit, Nir (25 July 2004). "On the inherent weakness of conditional synchronization primitives". *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing*. Association for Computing Machinery: 80–87. CiteSeerX 10.1.1.96.9340 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.96.9340). doi:10.1145/1011767.1011780 (https://doi.org/10.1145%2F1011767.1011780). ISBN 1581138024. S2CID 9313205 (https://api.semanticscholar.org/CorpusID:9313205).

Further reading

- Herlihy, M.; Shavit, N. (1999). "The topological structure of asynchronous computability". *Journal of the ACM*. **46** (6): 858. CiteSeerX 10.1.1.78.1455 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.1455). doi:10.1145/331524.331529 (https://doi.org/10.1145%2F331524.331529). S2CID 5797174 (https://api.semanticscholar.org/CorpusID:5797174).
- Saks, M.; Zaharoglou, F. (2000). "Wait-Free k-Set Agreement is Impossible: The Topology of Public Knowledge". *SIAM Journal on Computing*. **29** (5): 1449–1483. doi:10.1137/S0097539796307698 (https://doi.org/10.1137%2FS0097539796307698).

Distributed computing

Distributed computing is a field of computer science that studies distributed systems. A *distributed system* is a system whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system.[1][2] The components interact with one another in order to achieve a common goal. Three significant challenges of distributed systems are: maintaining concurrency of components, overcoming the lack of a global clock, and managing the independent failure of components.[1] When a component of one system fails, the entire system does not fail.[3] Examples of distributed systems vary from SOA-based systems to massively multiplayer online games to peer-to-peer applications.

A computer program that runs within a distributed system is called a **distributed program**,[4] and *distributed programming* is the process of writing such programs.[5] There are many different types of implementations for the message passing mechanism, including pure HTTP, RPC-like connectors and message queues.[6]

Distributed computing also refers to the use of distributed systems to solve computational problems. In *distributed computing*, a problem is divided into many tasks, each of which is solved by one or more computers,[7] which communicate with each other via message passing.[8]

Contents

Introduction

The word *distributed* in terms such as "distributed system", "distributed programming", and "distributed algorithm" originally referred to computer networks where individual computers were physically distributed within some geographical area.[9] The terms are nowadays used in a much wider sense, even referring to autonomous processes that run on the same physical computer and interact with each other by message passing.[8]

While there is no single definition of a distributed system,[10] the following defining properties are commonly used as:

- There are several autonomous computational entities (*computers* or *nodes*), each of which has its own local memory.[11]
- The entities communicate with each other by message passing.[12]

A distributed system may have a common goal, such as solving a large computational problem;[13] the user then perceives the collection of autonomous processors as a unit. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users.[14]

Other typical properties of distributed systems include the following:

- The system has to tolerate failures in individual computers.[15]
- The structure of the system (network topology, network latency, number of computers) is not known in advance, the system may consist of different kinds of computers and network links, and the system may change during the execution of a distributed program.[16]
- Each computer has only a limited, incomplete view of the system. Each computer may know only one part of the input.[17]

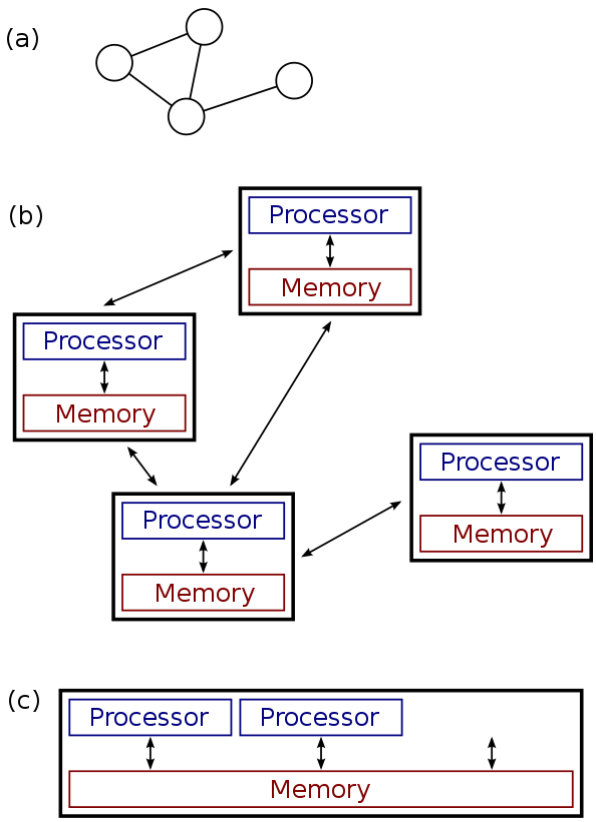
Parallel and distributed computing

Distributed systems are groups of networked computers which share a common goal for their work. The terms "concurrent computing", "parallel computing", and "distributed computing" have much overlap, and no clear distinction exists between them.[18] The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel.[19] Parallel computing may be seen as a particular tightly coupled form of distributed computing,[20] and distributed computing may be seen as a loosely coupled form of parallel computing.[10] Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors may have access to a shared memory to exchange information between processors.[21]
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.[22]

The figure on the right illustrates the difference between distributed and parallel systems. Figure (a) is a schematic view of a typical distributed system; the system is represented as a network topology in which each node is a computer and each line connecting the nodes is a communication link. Figure (b) shows the same distributed system in more detail: each computer has its own local memory, and information can be exchanged only by passing messages from one node to another by using the available communication links. Figure (c) shows a parallel system in which each processor has a direct access to a shared memory.

The situation is further complicated by the traditional uses of the terms parallel and distributed *algorithm* that do not quite match the above definitions of parallel and distributed *systems* (see below for more detailed discussion). Nevertheless, as a rule of thumb, high-performance parallel computation in a shared-memory multiprocessor uses parallel algorithms while the coordination of a large-scale distributed system uses distributed algorithms.[23]



(a), (b): a distributed system.
(c): a parallel system.

History

The use of concurrent processes which communicate through message-passing has its roots in operating system architectures studied in the 1960s.[24] The first widespread distributed systems were local-area networks such as Ethernet, which was invented in the 1970s.[25]

ARPANET, one of the predecessors of the Internet, was introduced in the late 1960s, and ARPANET e-mail was invented in the early 1970s. E-mail became the most successful application of ARPANET,[26] and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET (and its successor, the global Internet), other early worldwide computer networks included Usenet and FidoNet from the 1980s, both of which were used to support distributed discussion systems.[27]

The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s. The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its counterpart International Symposium on Distributed Computing (DISC) was first held in Ottawa in 1985 as the International Workshop on Distributed Algorithms on Graphs.[28]

Architectures

Various hardware and software architectures are used for distributed computing. At a lower level, it is necessary to interconnect multiple CPUs with some sort of network, regardless of whether that network is printed onto a circuit board or made up of loosely coupled devices and cables. At a higher level, it is necessary to interconnect processes running on those CPUs with some sort of communication system.[29]

Distributed programming typically falls into one of several basic architectures: client–server, three-tier, *n*-tier, or peer-to-peer; or categories: loose coupling, or tight coupling.[30]

- Client–server: architectures where smart clients contact the server for data then format and display it to the users. Input at the client is committed back to the server when it represents a permanent change.
- Three-tier: architectures that move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.

- *n*-tier: architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.
- Peer-to-peer: architectures where there are no special machines that provide a service or manage the network resources.[31]:227 Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and as servers.[32] Examples of this architecture include BitTorrent and the bitcoin network.

Another basic aspect of distributed computing architecture is the method of communicating and coordinating work among concurrent processes. Through various message passing protocols, processes may communicate directly with one another, typically in a master/slave relationship. Alternatively, a "database-centric" architecture can enable distributed computing to be done without any form of direct inter-process communication, by utilizing a shared database.[33] Database-centric architecture in particular provides relational processing analytics in a schematic architecture allowing for live environment relay. This enables distributed computing functions both within and beyond the parameters of a networked database.[34]

Applications

Reasons for using distributed systems and distributed computing may include:

- The very nature of an application may *require* the use of a communication network that connects several computers: for example, data produced in one physical location and required in another location.
- There are many cases in which the use of a single computer would be possible in principle, but the use of a distributed system is *beneficial* for practical reasons. For example:
  - It can allow for much larger storage and memory, faster compute, and higher bandwidth than a single machine.
  - It can provide more reliability than a non-distributed system, as there is no single point of failure. Moreover, a distributed system may be easier to expand and manage than a monolithic uniprocessor system.[35]
  - It may be more cost-efficient to obtain the desired level of performance by using a cluster of several low-end computers, in comparison with a single high-end computer.

Examples

Examples of distributed systems and applications of distributed computing include the following:[36]

- telecommunication networks:
  - telephone networks and cellular networks,
  - computer networks such as the Internet,
  - wireless sensor networks,
  - routing algorithms;
- network applications:
  - World Wide Web and peer-to-peer networks,
  - massively multiplayer online games and virtual reality communities,
  - distributed databases and distributed database management systems,
  - network file systems,
  - distributed cache such as burst buffers,

    - distributed information processing systems such as banking systems and airline reservation systems;
- real-time process control:
    - aircraft control systems,
    - industrial control systems;
- parallel computation:
    - scientific computing, including cluster computing, grid computing, cloud computing,[37] and various volunteer computing projects (see the list of distributed computing projects),
    - distributed rendering in computer graphics.
- peer-to-peer

Theoretical foundations

Models

Many tasks that we would like to automate by using a computer are of question–answer type: we would like to ask a question and the computer should produce an answer. In theoretical computer science, such tasks are called computational problems. Formally, a computational problem consists of *instances* together with a *solution* for each instance. Instances are questions that we can ask, and solutions are desired answers to these questions.

Theoretical computer science seeks to understand which computational problems can be solved by using a computer (computability theory) and how efficiently (computational complexity theory). Traditionally, it is said that a problem can be solved by using a computer if we can design an algorithm that produces a correct solution for any given instance. Such an algorithm can be implemented as a computer program that runs on a general-purpose computer: the program reads a problem instance from input, performs some computation, and produces the solution as output. Formalisms such as random-access machines or universal Turing machines can be used as abstract models of a sequential general-purpose computer executing such an algorithm.[38][39]

The field of concurrent and distributed computing studies similar questions in the case of either multiple computers, or a computer that executes a network of interacting processes: which computational problems can be solved in such a network and how efficiently? However, it is not at all obvious what is meant by "solving a problem" in the case of a concurrent or distributed system: for example, what is the task of the algorithm designer, and what is the concurrent or distributed equivalent of a sequential general-purpose computer?

The discussion below focuses on the case of multiple computers, although many of the issues are the same for concurrent processes running on a single computer.

Three viewpoints are commonly used:

Parallel algorithms in shared-memory model

- All processors have access to a shared memory. The algorithm designer chooses the program executed by each processor.
- One theoretical model is the parallel random-access machines (PRAM) that are used.[40] However, the classical PRAM model assumes synchronous access to the shared memory.
- Shared-memory programs can be extended to distributed systems if the underlying operating system encapsulates the communication between nodes and virtually unifies the memory across all individual systems.

- A model that is closer to the behavior of real-world multiprocessor machines and takes into account the use of machine instructions, such as Compare-and-swap (CAS), is that of *asynchronous shared memory*. There is a wide body of work on this model, a summary of which can be found in the literature.[41][42]

Parallel algorithms in message-passing model

- The algorithm designer chooses the structure of the network, as well as the program executed by each computer.
- Models such as Boolean circuits and sorting networks are used.[43] A Boolean circuit can be seen as a computer network: each gate is a computer that runs an extremely simple computer program. Similarly, a sorting network can be seen as a computer network: each comparator is a computer.

Distributed algorithms in message-passing model

- The algorithm designer only chooses the computer program. All computers run the same program. The system must work correctly regardless of the structure of the network.
- A commonly used model is a graph with one finite-state machine per node.

In the case of distributed algorithms, computational problems are typically related to graphs. Often the graph that describes the structure of the computer network *is* the problem instance. This is illustrated in the following example.

An example

Consider the computational problem of finding a coloring of a given graph *G*. Different fields might take the following approaches:

Centralized algorithms

- The graph *G* is encoded as a string, and the string is given as input to a computer. The computer program finds a coloring of the graph, encodes the coloring as a string, and outputs the result.

Parallel algorithms

- Again, the graph *G* is encoded as a string. However, multiple computers can access the same string in parallel. Each computer might focus on one part of the graph and produce a coloring for that part.
- The main focus is on high-performance computation that exploits the processing power of multiple computers in parallel.

Distributed algorithms

- The graph *G* is the structure of the computer network. There is one computer for each node of *G* and one communication link for each edge of *G*. Initially, each computer only knows about its immediate neighbors in the graph *G*; the computers must exchange messages with each other to discover more about the structure of *G*. Each computer must produce its own color as output.
- The main focus is on coordinating the operation of an arbitrary distributed system.

While the field of parallel algorithms has a different focus than the field of distributed algorithms, there is much interaction between the two fields. For example, the Cole–Vishkin algorithm for graph coloring[44] was originally presented as a parallel algorithm, but the same technique can also

be used directly as a distributed algorithm.

Moreover, a parallel algorithm can be implemented either in a parallel system (using shared memory) or in a distributed system (using message passing).[45] The traditional boundary between parallel and distributed algorithms (choose a suitable network vs. run in any given network) does not lie in the same place as the boundary between parallel and distributed systems (shared memory vs. message passing).

Complexity measures

In parallel algorithms, yet another resource in addition to time and space is the number of computers. Indeed, often there is a trade-off between the running time and the number of computers: the problem can be solved faster if there are more computers running in parallel (see speedup). If a decision problem can be solved in polylogarithmic time by using a polynomial number of processors, then the problem is said to be in the class NC.[46] The class NC can be defined equally well by using the PRAM formalism or Boolean circuits—PRAM machines can simulate Boolean circuits efficiently and vice versa.[47]

In the analysis of distributed algorithms, more attention is usually paid on communication operations than computational steps. Perhaps the simplest model of distributed computing is a synchronous system where all nodes operate in a lockstep fashion. This model is commonly known as the LOCAL model. During each *communication round*, all nodes in parallel (1) receive the latest messages from their neighbours, (2) perform arbitrary local computation, and (3) send new messages to their neighbors. In such systems, a central complexity measure is the number of synchronous communication rounds required to complete the task.[48]

This complexity measure is closely related to the diameter of the network. Let D be the diameter of the network. On the one hand, any computable problem can be solved trivially in a synchronous distributed system in approximately $2D$ communication rounds: simply gather all information in one location (D rounds), solve the problem, and inform each node about the solution (D rounds).

On the other hand, if the running time of the algorithm is much smaller than D communication rounds, then the nodes in the network must produce their output without having the possibility to obtain information about distant parts of the network. In other words, the nodes must make globally consistent decisions based on information that is available in their *local D-neighbourhood*. Many distributed algorithms are known with the running time much smaller than D rounds, and understanding which problems can be solved by such algorithms is one of the central research questions of the field.[49] Typically an algorithm which solves a problem in polylogarithmic time in the network size is considered efficient in this model.

Another commonly used measure is the total number of bits transmitted in the network (cf. communication complexity).[50] The features of this concept are typically captured with the CONGEST(B) model, which is similarly defined as the LOCAL model, but where single messages can only contain B bits.

Other problems

Traditional computational problems take the perspective that the user asks a question, a computer (or a distributed system) processes the question, then produces an answer and stops. However, there are also problems where the system is required not to stop, including the dining philosophers problem and other similar mutual exclusion problems. In these problems, the distributed system is supposed to continuously coordinate the use of shared resources so that no conflicts or deadlocks occur.

There are also fundamental challenges that are unique to distributed computing, for example those related to *fault-tolerance*. Examples of related problems include consensus problems,[51] Byzantine fault tolerance,[52] and self-stabilisation.[53]

Much research is also focused on understanding the *asynchronous* nature of distributed systems:

- Synchronizers can be used to run synchronous algorithms in asynchronous systems.[54]
- Logical clocks provide a causal happened-before ordering of events.[55]
- Clock synchronization algorithms provide globally consistent physical time stamps.[56]

Election

Coordinator election (or *leader election*) is the process of designating a single process as the organizer of some task distributed among several computers (nodes). Before the task is begun, all network nodes are either unaware which node will serve as the "coordinator" (or leader) of the task, or unable to communicate with the current coordinator. After a coordinator election algorithm has been run, however, each node throughout the network recognizes a particular, unique node as the task coordinator.[57]

The network nodes communicate among themselves in order to decide which of them will get into the "coordinator" state. For that, they need some method in order to break the symmetry among them. For example, if each node has unique and comparable identities, then the nodes can compare their identities, and decide that the node with the highest identity is the coordinator.[57]

The definition of this problem is often attributed to LeLann, who formalized it as a method to create a new token in a token ring network in which the token has been lost.[58]

Coordinator election algorithms are designed to be economical in terms of total bytes transmitted, and time. The algorithm suggested by Gallager, Humblet, and Spira [59] for general undirected graphs has had a strong impact on the design of distributed algorithms in general, and won the Dijkstra Prize for an influential paper in distributed computing.

Many other algorithms were suggested for different kinds of network graphs, such as undirected rings, unidirectional rings, complete graphs, grids, directed Euler graphs, and others. A general method that decouples the issue of the graph family from the design of the coordinator election algorithm was suggested by Korach, Kutten, and Moran.[60]

In order to perform coordination, distributed systems employ the concept of coordinators. The coordinator election problem is to choose a process from among a group of processes on different processors in a distributed system to act as the central coordinator. Several central coordinator election algorithms exist.[61]

Properties of distributed systems

So far the focus has been on *designing* a distributed system that solves a given problem. A complementary research problem is *studying* the properties of a given distributed system.[62][63]

The halting problem is an analogous example from the field of centralised computation: we are given a computer program and the task is to decide whether it halts or runs forever. The halting problem is undecidable in the general case, and naturally understanding the behaviour of a computer network is at least as hard as understanding the behaviour of one computer.[64]

However, there are many interesting special cases that are decidable. In particular, it is possible to reason about the behaviour of a network of finite-state machines. One example is telling whether a given network of interacting (asynchronous and non-deterministic) finite-state machines can reach a deadlock. This problem is PSPACE-complete,[65] i.e., it is decidable, but not likely that there is an efficient (centralised, parallel or distributed) algorithm that solves the problem in the case of large networks.

See also

- Dataflow programming
- Distributed networking
- Decentralized computing
- Federation (information technology)
- AppScale
- BOINC
- Code mobility
- Distributed algorithm
- Distributed algorithmic mechanism design
- Distributed cache
- Distributed operating system
- Edsger W. Dijkstra Prize in Distributed Computing
- Fog computing
- Folding@home
- Grid computing
- Inferno
- Jungle computing
- Layered queueing network
- Library Oriented Architecture (LOA)
- List of distributed computing conferences
- List of distributed computing projects
- List of important publications in concurrent, parallel, and distributed computing
- Model checking
- Parallel distributed processing
- Parallel programming model
- Plan 9 from Bell Labs
- Shared nothing architecture

Notes

1. Tanenbaum, Andrew S.; Steen, Maarten van (2002). *Distributed systems: principles and paradigms* (https://www.distributed-systems.net/index.php/books/ds3/). Upper Saddle River, NJ: Pearson Prentice Hall. ISBN 0-13-088893-1.
2. "Distributed Programs". *Texts in Computer Science*. London: Springer London. 2010. pp. 373–406. doi:10.1007/978-1-84882-745-5_11 (https://doi.org/10.1007%2F978-1-84882-745-5_11). ISBN 978-1-84882-744-8. ISSN 1868-0941 (https://www.worldcat.org/issn/1868-0941). "Systems consist of a number of physically distributed components that work independently using their private storage, but also communicate from time to time by explicit message passing. Such systems are called distributed systems."
3. Dusseau & Dusseau 2016, p. 1-2.
4. "Distributed Programs". *Texts in Computer Science*. London: Springer London. 2010. pp. 373–406. doi:10.1007/978-1-84882-745-5_11 (https://doi.org/10.1007%2F978-1-84882-745-5_11). ISBN 978-1-84882-744-8. ISSN 1868-0941 (https://www.worldcat.org/issn/1868-0941). "Distributed programs are abstract descriptions of distributed systems. A distributed program consists of a collection of processes that work concurrently and communicate by explicit message passing. Each process can access a set of variables which are disjoint from the variables that can be changed by any other process."
5. Andrews (2000). Dolev (2000). Ghosh (2007), p. 10.
6. Magnoni, L. (2015). "Modern Messaging for Distributed Sytems (sic)" (https://doi.org/10.1088%2F1742-6596%2F608%2F1%2F012038). *Journal of Physics: Conference Series*. **608** (1): 012038. doi:10.1088/1742-6596/608/1/012038 (https://doi.org/10.1088%2F1742-6596%2F608%2F1%2F012038). ISSN 1742-6596 (https://www.worldcat.org/issn/1742-6596).
7. Godfrey (2002).
8. Andrews (2000), p. 291–292. Dolev (2000), p. 5.

9. Lynch (1996), p. 1.
10. Ghosh (2007), p. 10.
11. Andrews (2000), pp. 8–9, 291. Dolev (2000), p. 5. Ghosh (2007), p. 3. Lynch (1996), p. xix, 1. Peleg (2000), p. xv.
12. Andrews (2000), p. 291. Ghosh (2007), p. 3. Peleg (2000), p. 4.
13. Ghosh (2007), p. 3–4. Peleg (2000), p. 1.
14. Ghosh (2007), p. 4. Peleg (2000), p. 2.
15. Ghosh (2007), p. 4, 8. Lynch (1996), p. 2–3. Peleg (2000), p. 4.
16. Lynch (1996), p. 2. Peleg (2000), p. 1.
17. Ghosh (2007), p. 7. Lynch (1996), p. xix, 2. Peleg (2000), p. 4.
18. Ghosh (2007), p. 10. Keidar (2008).
19. Lynch (1996), p. xix, 1–2. Peleg (2000), p. 1.
20. Peleg (2000), p. 1.
21. Papadimitriou (1994), Chapter 15. Keidar (2008).
22. See references in Introduction.
23. Bentaleb, A.; Yifan, L.; Xin, J.; et al. (2016). "Parallel and Distributed Algorithms" (http://www.comp.nus.edu.sg/~rahul/allfiles/cs6234-16-pds.pdf) (PDF). National University of Singapore. Retrieved 20 July 2018.
24. Andrews (2000), p. 348.
25. Andrews (2000), p. 32.
26. Peter (2004), The history of email (http://www.nethistory.info/History%20of%20the%20Internet/email.html).
27. Banks, M. (2012). *On the Way to the Web: The Secret History of the Internet and its Founders* (https://books.google.com/books?id=1J78hiHKaPoC&pg=PT67). Apress. pp. 44–5. ISBN 9781430250746.
28. Tel, G. (2000). *Introduction to Distributed Algorithms* (https://books.google.com/books?id=vlpnS25qAJQC&pg=PA35). Cambridge University Press. pp. 35–36. ISBN 9780521794831.
29. Ohlídal, M.; Jaroš, J.; Schwarz, J.; et al. (2006). "Evolutionary Design of OAB and AAB Communication Schedules for Interconnection Networks". In Rothlauf, F.; Branke, J.; Cagnoni, S. (eds.). *Applications of Evolutionary Computing*. Springer Science & Business Media. pp. 267–78. ISBN 9783540332374.
30. "Real Time And Distributed Computing Systems" (https://web.archive.org/web/20170110015222/https://pdfs.semanticscholar.org/2950/37ee46ac281590f67435380cebc385ac9749.pdf) (PDF). ISSN 2278-0661 (https://www.worldcat.org/issn/2278-0661). Archived from the original (https://pdfs.semanticscholar.org/2950/37ee46ac281590f67435380cebc385ac9749.pdf) (PDF) on 2017-01-10. Retrieved 2017-01-09.
31. Vigna P, Casey MJ. *The Age of Cryptocurrency: How Bitcoin and the Blockchain Are Challenging the Global Economic Order* St. Martin's Press January 27, 2015 ISBN 9781250065636
32. Hieu., Vu, Quang (2010). *Peer-to-peer computing : principles and applications*. Lupu, Mihai., Ooi, Beng Chin, 1961-. Heidelberg: Springer. p. 16. ISBN 9783642035135. OCLC 663093862 (https://www.worldcat.org/oclc/663093862).
33. Lind P, Alm M (2006), "A database-centric virtual chemistry system", *J Chem Inf Model*, **46** (3): 1034–9, doi:10.1021/ci050360b (https://doi.org/10.1021%2Fci050360b), PMID 16711722 (https://pubmed.ncbi.nlm.nih.gov/16711722).
34. Chiu, G (1990). "A model for optimal database allocation in distributed computing systems". *Proceedings. IEEE INFOCOM'90: Ninth Annual Joint Conference of the IEEE Computer and Communications Societies*.
35. Elmasri & Navathe (2000), Section 24.1.2.

36. Andrews (2000), p. 10–11. Ghosh (2007), p. 4–6. Lynch (1996), p. xix, 1. Peleg (2000), p. xv. Elmasri & Navathe (2000), Section 24.
37. Haussmann, J. (2019). "Cost-efficient parallel processing of irregularly structured problems in cloud computing environments". *Journal of Cluster Computing*. **22** (3): 887–909. doi:10.1007/s10586-018-2879-3 (https://doi.org/10.1007%2Fs10586-018-2879-3). S2CID 54447518 (https://api.semanticscholar.org/CorpusID:54447518).
38. Toomarian, N.B.; Barhen, J.; Gulati, S. (1992). "Neural Networks for Real-Time Robotic Applications" (https://books.google.com/books?id=CKTsCgAAQBAJ&pg=PA214). In Fijany, A.; Bejczy, A. (eds.). *Parallel Computation Systems For Robotics: Algorithms And Architectures*. World Scientific. p. 214. ISBN 9789814506175.
39. Savage, J.E. (1998). *Models of Computation: Exploring the Power of Computing*. Addison Wesley. p. 209. ISBN 9780201895391.
40. Cormen, Leiserson & Rivest (1990), Section 30.
41. Herlihy & Shavit (2008), Chapters 2-6.
42. Lynch (1996)
43. Cormen, Leiserson & Rivest (1990), Sections 28 and 29.
44. Cole & Vishkin (1986). Cormen, Leiserson & Rivest (1990), Section 30.5.
45. Andrews (2000), p. ix.
46. Arora & Barak (2009), Section 6.7. Papadimitriou (1994), Section 15.3.
47. Papadimitriou (1994), Section 15.2.
48. Lynch (1996), p. 17–23.
49. Peleg (2000), Sections 2.3 and 7. Linial (1992). Naor & Stockmeyer (1995).
50. Schneider, J.; Wattenhofer, R. (2011). "Trading Bit, Message, and Time Complexity of Distributed Algorithms" (https://books.google.com/books?id=dT6nwpXvES4C&pg=PA51). In Peleg, D. (ed.). *Distributed Computing*. Springer Science & Business Media. pp. 51–65. ISBN 9783642240997.
51. Lynch (1996), Sections 5–7. Ghosh (2007), Chapter 13.
52. Lynch (1996), p. 99–102. Ghosh (2007), p. 192–193.
53. Dolev (2000). Ghosh (2007), Chapter 17.
54. Lynch (1996), Section 16. Peleg (2000), Section 6.
55. Lynch (1996), Section 18. Ghosh (2007), Sections 6.2–6.3.
56. Ghosh (2007), Section 6.4.
57. Haloi, S. (2015). *Apache ZooKeeper Essentials* (https://books.google.com/books?id=Ym9uBgAAQBAJ&pg=PA100). Packt Publishing Ltd. pp. 100–101. ISBN 9781784398323.
58. LeLann, G. (1977). "Distributed systems - toward a formal approach". *Information Processing*. **77**: 155·160 – via Elsevier.
59. R. G. Gallager, P. A. Humblet, and P. M. Spira (January 1983). "A Distributed Algorithm for Minimum-Weight Spanning Trees" (http://www.apposite-tech.com/blog/wp-content/uploads/2017/09/p66-gallager.pdf) (PDF). *ACM Transactions on Programming Languages and Systems*. **5** (1): 66–77. doi:10.1145/357195.357200 (https://doi.org/10.1145%2F357195.357200). S2CID 2758285 (https://api.semanticscholar.org/CorpusID:2758285).
60. Korach, Ephraim; Kutten, Shay; Moran, Shlomo (1990). "A Modular Technique for the Design of Efficient Distributed Leader Finding Algorithms" (https://www.cs.technion.ac.il/~moran/r/PS/kkm.pdf) (PDF). *ACM Transactions on Programming Languages and Systems*. **12** (1): 84–101. CiteSeerX 10.1.1.139.7342 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.139.7342). doi:10.1145/77606.77610 (https://doi.org/10.1145%2F77606.77610). S2CID 9175968 (https://api.semanticscholar.org/CorpusID:9175968).
61. Hamilton, Howard. "Distributed Algorithms" (http://www2.cs.uregina.ca/~hamilton/courses/330/notes/distributed/distributed.html). Retrieved 2013-03-03.

62. "Major unsolved problems in distributed systems?" (https://cstheory.stackexchange.com/q/10045). *cstheory.stackexchange.com*. Retrieved 16 March 2018.
63. "How big data and distributed systems solve traditional scalability problems" (http://www.theserverside.com/feature/How-big-data-and-distributed-systems-solve-traditional-scalability-problems). *theserverside.com*. Retrieved 16 March 2018.
64. Svozil, K. (2011). "Indeterminism and Randomness Through Physics" (https://books.google.com/books?id=ep_FCgAAQBAJ&pg=PA112). In Hector, Z. (ed.). *Randomness Through Computation: Some Answers, More Questions*. World Scientific. pp. 112–3. ISBN 9789814462631.
65. Papadimitriou (1994), Section 19.3.

References

Books

- Andrews, Gregory R. (2000), *Foundations of Multithreaded, Parallel, and Distributed Programming* (https://archive.org/details/foundationsofmul0000andr), Addison–Wesley, ISBN 978-0-201-35752-3.
- Arora, Sanjeev; Barak, Boaz (2009), *Computational Complexity – A Modern Approach*, Cambridge, ISBN 978-0-521-42426-4.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990), *Introduction to Algorithms* (1st ed.), MIT Press, ISBN 978-0-262-03141-7.
- Dolev, Shlomi (2000), *Self-Stabilization*, MIT Press, ISBN 978-0-262-04178-2.
- Elmasri, Ramez; Navathe, Shamkant B. (2000), *Fundamentals of Database Systems* (3rd ed.), Addison–Wesley, ISBN 978-0-201-54263-9.
- Ghosh, Sukumar (2007), *Distributed Systems – An Algorithmic Approach*, Chapman & Hall/CRC, ISBN 978-1-58488-564-1.
- Lynch, Nancy A. (1996), *Distributed Algorithms* (https://archive.org/details/distributedalgor0000lync), Morgan Kaufmann, ISBN 978-1-55860-348-6.
- Herlihy, Maurice P.; Shavit, Nir N. (2008), *The Art of Multiprocessor Programming*, Morgan Kaufmann, ISBN 978-0-12-370591-4.
- Papadimitriou, Christos H. (1994), *Computational Complexity*, Addison–Wesley, ISBN 978-0-201-53082-7.
- Peleg, David (2000), *Distributed Computing: A Locality-Sensitive Approach* (https://web.archive.org/web/20090806070332/http://www.ec-securehost.com/SIAM/DT05.html), SIAM, ISBN 978-0-89871-464-7, archived from the original (http://www.ec-securehost.com/SIAM/DT05.html) on 2009-08-06, retrieved 2009-07-16.

Articles

- Cole, Richard; Vishkin, Uzi (1986), "Deterministic coin tossing with applications to optimal parallel list ranking", *Information and Control*, **70** (1): 32–53, doi:10.1016/S0019-9958(86)80023-7 (https://doi.org/10.1016%2FS0019-9958%2886%2980023-7).
- Keidar, Idit (2008), "Distributed computing column 32 – The year in review" (http://webee.technion.ac.il/~idish/sigactNews/#column%2032), *ACM SIGACT News*, **39** (4): 53–54, CiteSeerX 10.1.1.116.1285 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.116.1285), doi:10.1145/1466390.1466402 (https://doi.org/10.1145%2F1466390.1466402), S2CID 7607391 (https://api.semanticscholar.org/CorpusID:7607391).
- Linial, Nathan (1992), "Locality in distributed graph algorithms", *SIAM Journal on Computing*, **21** (1): 193–201, CiteSeerX 10.1.1.471.6378 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.471.6378), doi:10.1137/0221015 (https://doi.org/10.1137%2F0221015).

- Naor, Moni; Stockmeyer, Larry (1995), "What can be computed locally?" (http://www.wisdom.weizmann.ac.il/~naor/PAPERS/lcl.pdf) (PDF), *SIAM Journal on Computing*, **24** (6): 1259–1277, CiteSeerX 10.1.1.29.669 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.669), doi:10.1137/S0097539793254571 (https://doi.org/10.1137%2FS0097539793254571).

Web sites

- Godfrey, Bill (2002). "A primer on distributed computing" (https://billpg.com/bacchae-co-uk/docs/dist.html).
- Peter, Ian (2004). "Ian Peter's History of the Internet" (http://www.nethistory.info/History%20of%20the%20Internet/). Retrieved 2009-08-04.

Further reading

Books

- Attiya, Hagit and Jennifer Welch (2004), *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*, Wiley-Interscience ISBN 0-471-45324-2.
- Christian Cachin; Rachid Guerraoui; Luís Rodrigues (2011), *Introduction to Reliable and Secure Distributed Programming* (2. ed.), Springer, Bibcode:2011itra.book.....C (https://ui.adsabs.harvard.edu/abs/2011itra.book.....C), ISBN 978-3-642-15259-7
- Coulouris, George; et al. (2011), *Distributed Systems: Concepts and Design (5th Edition)*, Addison-Wesley ISBN 0-132-14301-1.
- Faber, Jim (1998), *Java Distributed Computing* (http://docstore.mik.ua/orelly/java-ent/dist/index.htm), O'Reilly: Java Distributed Computing by Jim Faber, 1998 (http://docstore.mik.ua/orelly/java-ent/dist/index.htm)
- Garg, Vijay K. (2002), *Elements of Distributed Computing*, Wiley-IEEE Press ISBN 0-471-03600-5.
- Tel, Gerard (1994), *Introduction to Distributed Algorithms*, Cambridge University Press
- Chandy, Mani; et al. (1988), *Parallel Program Design*, Addison-Wesley ISBN 0201058669
- Dusseau, Remzi H.; Dusseau, Andrea (2016). *Operating Systems: Three Easy Pieces, Chapter 48 Distributed Systems* (https://web.archive.org/web/20210831013525/https://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf) (PDF). Archived from the original (https://pages.cs.wisc.edu/~remzi/OSTEP/dist-intro.pdf) (PDF) on 31 August 2021. Retrieved 8 October 2021.

Articles

- Keidar, Idit; Rajsbaum, Sergio, eds. (2000–2009), "Distributed computing column" (http://webee.technion.ac.il/~idish/sigactNews/), *ACM SIGACT News*.
- Birrell, A. D.; Levin, R.; Schroeder, M. D.; Needham, R. M. (April 1982). "Grapevine: An exercise in distributed computing" (http://web.cs.wpi.edu/~cs4513/d07/Papers/Birrell,%20Levin,%20et.%20al.,%20Grapevine.pdf) (PDF). *Communications of the ACM*. **25** (4): 260–274. doi:10.1145/358468.358487 (https://doi.org/10.1145%2F358468.358487). S2CID 16066616 (https://api.semanticscholar.org/CorpusID:16066616).

Conference Papers

- Rodriguez, Carlos; Villagra, Marcos; Baran, Benjamin (2007). "Asynchronous team algorithms for Boolean Satisfiability". *2007 2nd Bio-Inspired Models of Network, Information and Computing Systems*. pp. 66–69. doi:10.1109/BIMNICS.2007.4610083 (https://doi.org/10.1109%2FBIMNICS.2007.4610083). S2CID 15185219 (https://api.semanticscholar.org/CorpusID:15185219).

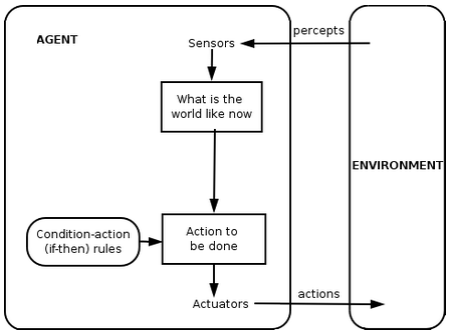
External links

- Media related to Distributed computing at Wikimedia Commons
- Distributed computing (https://curlie.org/Computers/Computer_Science/Distributed_Computing/) at Curlie
- Distributed computing journals (https://curlie.org/Computers/Computer_Science/Distributed_Computing/Publications/) at Curlie

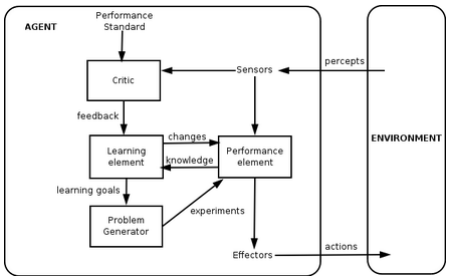
---

Multi-agent system

A **multi-agent system** (**MAS** or "self-organized system") is a computerized system composed of multiple interacting intelligent agents.[1] Multi-agent systems can solve problems that are difficult or impossible for an individual agent or a monolithic system to solve.[2] Intelligence may include methodic, functional, procedural approaches, algorithmic search or reinforcement learning.[3][4]



Simple reflex agent



Learning agent

Despite considerable overlap, a multi-agent system is not always the same as an agent-based model (ABM). The goal of an ABM is to search for explanatory insight into the collective behavior of agents (which don't necessarily need to be "intelligent") obeying simple rules, typically in natural systems, rather than in solving specific practical or engineering problems. The terminology of ABM tends to be used more often in the science, and MAS in engineering and technology.[5] Applications where multi-agent systems research may deliver an appropriate approach include online trading,[6] disaster response,[7][8] target surveillance [9] and social structure modelling.[10]

Contents

Concept

Multi-agent systems consist of agents and their environment. Typically multi-agent systems research refers to software agents. However, the agents in a multi-agent system could equally well be robots, humans or human teams. A multi-agent system may contain combined human-agent teams.

Agents can be divided into types spanning simple to complex. Categories include:

- Passive agents[11] or "agent without goals" (such as obstacle, apple or key in any simple simulation)
- Active agents[11] with simple goals (like birds in flocking, or wolf–sheep in prey-predator model)
- Cognitive agents (complex calculations)

Agent environments can be divided into:

- Virtual
- Discrete
- Continuous

Agent environments can also be organized according to properties such as accessibility (whether it is possible to gather complete information about the environment), determinism (whether an action causes a definite effect), dynamics (how many entities influence the environment in the moment), discreteness (whether the number of possible actions in the environment is finite), episodicity (whether agent actions in certain time periods influence other periods),[12] and dimensionality (whether spatial characteristics are important factors of the environment and the agent considers space in its decision making).[13] Agent actions are typically mediated via an appropriate middleware. This middleware offers a first-class design abstraction for multi-agent systems, providing means to govern resource access and agent coordination.[14]

Characteristics

The agents in a multi-agent system have several important characteristics:[15]

- Autonomy: agents at least partially independent, self-aware, autonomous
- Local views: no agent has a full global view, or the system is too complex for an agent to exploit such knowledge
- Decentralization: no agent is designated as controlling (or the system is effectively reduced to a monolithic system)[16]

Self-organisation and self-direction

Multi-agent systems can manifest self-organisation as well as self-direction and other control paradigms and related complex behaviors even when the individual strategies of all their agents are simple. When agents can share knowledge using any agreed language, within the constraints of the system's communication protocol, the approach may lead to a common improvement. Example languages are Knowledge Query Manipulation Language (KQML) or Agent Communication Language (ACL).

System paradigms

Many MAS are implemented in computer simulations, stepping the system through discrete "time steps". The MAS components communicate typically using a weighted request matrix, e.g.

```
Speed-VERY_IMPORTANT: min=45 mph,
Path length-MEDIUM_IMPORTANCE: max=60 expectedMax=40,
Max-Weight-UNIMPORTANT
Contract Priority-REGULAR
```

and a weighted response matrix, e.g.

```
Speed-min:50 but only if weather sunny,
Path length:25 for sunny / 46 for rainy
Contract Priority-REGULAR
note – ambulance will override this priority and you'll have to wait
```

A challenge-response-contract scheme is common in MAS systems, where

- First a "Who can?" question is distributed.
- Only the relevant components respond: "I can, at this price".
- Finally, a contract is set up, usually in several short communication steps between sides,

also considering other components, evolving "contracts" and the restriction sets of the component algorithms.

Another paradigm commonly used with MAS is the "pheromone", where components leave information for other nearby components. These pheromones may evaporate/concentrate with time, that is their values may decrease (or increase).

Properties

MAS tend to find the best solution for their problems without intervention. There is high similarity here to physical phenomena, such as energy minimizing, where physical objects tend to reach the lowest energy possible within the physically constrained world. For example: many of the cars entering a metropolis in the morning will be available for leaving that same metropolis in the evening.

The systems also tend to prevent propagation of faults, self-recover and be fault tolerant, mainly due to the redundancy of components.

Research

The study of multi-agent systems is "concerned with the development and analysis of sophisticated AI problem-solving and control architectures for both single-agent and multiple-agent systems."[17] Research topics include:

- agent-oriented software engineering
- beliefs, desires, and intentions (BDI)
- cooperation and coordination
- distributed constraint optimization (DCOPs)
- organization
- communication
- negotiation
- distributed problem solving
- multi-agent learning[18]
- agent mining
- scientific communities (e.g., on biological flocking, language evolution, and economics)[19][20]
- dependability and fault-tolerance
- robotics,[21] multi-robot systems (MRS), robotic clusters

Frameworks

Frameworks have emerged that implement common standards (such as the FIPA and OMG MASIF[22] standards). These frameworks e.g. JADE, save time and aid in the standardization of MAS development.[23]

Currently though, no standard is actively maintained from FIPA or OMG. Efforts for further development of software agents in industrial context are carried out in IEEE IES technical committee on Industrial Agents.[24]

Applications

MAS have not only been applied in academic research, but also in industry.[25] MAS are applied in the real world to graphical applications such as computer games. Agent systems have been used in films.[26] It is widely advocated for use in networking and mobile technologies, to achieve automatic and dynamic load balancing, high scalability and self-healing networks. They are being used for coordinated defence systems.

Other applications[27] include transportation,[28] logistics,[29] graphics, manufacturing, power system,[30] smartgrids[31] and GIS.

Also, Multi-agent Systems Artificial Intelligence (MAAI) are used for simulating societies, the purpose thereof being helpful in the fields of climate, energy, epidemiology, conflict management, child abuse, ...[32] Some organisations working on using multi-agent system models include Center for Modelling Social Systems, Centre for Research in Social Simulation, Centre for Policy Modelling, Society for Modelling and Simulation International.[32] Hallerbach et al. discussed the application of agent-based approaches for the development and validation of automated driving systems via a digital twin of the vehicle-under-test and microscopic traffic simulation based on independent agents.[33] Waymo has created a multi-agent simulation environment Carcraft to test algorithms for self-driving cars.[34][35] It simulates traffic interactions between human drivers, pedestrians and automated vehicles. People's behavior is imitated by artificial agents based on data of real human behavior.

See also

- Comparison of agent-based modeling software
- Agent-based computational economics (ACE)
- Artificial brain
- Artificial intelligence
- Artificial life
- Artificial life framework
- AI mayor
- Black box
- Blackboard system
- Complex systems
- Discrete event simulation
- Distributed artificial intelligence
- Emergence
- Evolutionary computation
- Game theory
- Human-based genetic algorithm
- Knowledge Query and Manipulation Language (KQML)
- Microbial intelligence

- Multi-agent planning
- Multi-agent reinforcement learning
- Pattern-oriented modeling
- PlatBox Project
- Reinforcement learning
- Scientific community metaphor
- Self-reconfiguring modular robot
- Simulated reality
- Social simulation
- Software agent
- Swarm intelligence
- Swarm robotics

References

1. Hu, J.; Bhowmick, P.; Jang, I.; Arvin, F.; Lanzon, A., "A Decentralized Cluster Formation Containment Framework for Multirobot Systems (https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9423979)" IEEE Transactions on Robotics, 2021.
2. Hu, J.; Turgut, A.; Lennox, B.; Arvin, F., "Robust Formation Coordination of Robot Swarms with Nonlinear Dynamics and Unknown Disturbances: Design and Experiments (https://ieeexplore.ieee.org/abstract/document/9409965)" IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.
3. Hu, J.; Bhowmick, P.; Lanzon, A., "Group Coordinated Control of Networked Mobile Robots with Applications to Object Transportation (https://ieeexplore.ieee.org/document/9468402)" IEEE Transactions on Vehicular Technology, 2021.
4. Wiering, M. A. (2000). "Multi-agent reinforcement learning for traffic light control". *Machine Learning: Proceedings of the Seventeenth International Conference (Icml'2000)*: 1151–1158. hdl:1874/20827 (https://hdl.handle.net/1874%2F20827).
5. Niazi, Muaz; Hussain, Amir (2011). "Agent-based Computing from Multi-agent Systems to Agent-Based Models: A Visual Survey" (https://www.researchgate.net/publication/220365334) (PDF). *Scientometrics*. **89** (2): 479–499. arXiv:1708.05872 (https://arxiv.org/abs/1708.05872). doi:10.1007/s11192-011-0468-9 (https://doi.org/10.1007%2Fs11192-011-0468-9). S2CID 17934527 (https://api.semanticscholar.org/CorpusID:17934527).
6. Rogers, Alex; David, E.; Schiff, J.; Jennings, N.R. (2007). "The Effects of Proxy Bidding and Minimum Bid Increments within eBay Auctions" (http://eprints.ecs.soton.ac.uk/12716/). *ACM Transactions on the Web*. **1** (2): 9–es. CiteSeerX 10.1.1.65.4539 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.4539). doi:10.1145/1255438.1255441 (https://doi.org/10.1145%2F1255438.1255441). S2CID 207163424 (https://api.semanticscholar.org/CorpusID:207163424).
7. Schurr, Nathan; Marecki, Janusz; Tambe, Milind; Scerri, Paul; Kasinadhuni, Nikhil; Lewis, J.P. (2005). "The Future of Disaster Response: Humans Working with Multiagent Teams using DEFACTO" (http://teamcore.usc.edu/papers/2005/SS105SchurrN.pdf) (PDF).
8. Genc, Zulkuf; et al. (2013). "Agent-based information infrastructure for disaster management" (http://www.gdmc.nl/gi4dmdocs/Gi4DM_2012_Genc.pdf) (PDF). *Intelligent Systems for Crisis Management*. Lecture Notes in Geoinformation and Cartography: 349–355. doi:10.1007/978-3-642-33218-0_26 (https://doi.org/10.1007%2F978-3-642-33218-0_26). ISBN 978-3-642-33217-3.

9. Hu, Junyan; Bhowmick, Parijat; Lanzon, Alexander (2020). "Distributed Adaptive Time-Varying Group Formation Tracking for Multiagent Systems With Multiple Leaders on Directed Graphs" (https://doi.org/10.1109%2FTCNS.2019.2913619). *IEEE Transactions on Control of Network Systems*. **7**: 140–150. doi:10.1109/TCNS.2019.2913619 (https://doi.org/10.1109%2FTCNS.2019.2913619). S2CID 149609966 (https://api.semanticscholar.org/CorpusID:149609966).
10. Sun, Ron; Naveh, Isaac (30 June 2004). "Simulating Organizational Decision-Making Using a Cognitively Realistic Agent Model" (http://jasss.soc.surrey.ac.uk/7/3/5.html). *Journal of Artificial Societies and Social Simulation*.
11. Kubera, Yoann; Mathieu, Philippe; Picault, Sébastien (2010), "Everything can be Agent!" (http://www.lifl.fr/SMAC/publications/pdf/aamas2010-everything.pdf) (PDF), *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2010)*: 1547–1548
12. Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (http://aima.cs.berkeley.edu/) (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2
13. Salamon, Tomas (2011). *Design of Agent-Based Models* (http://www.designofagentbasedmodels.info/). Repin: Bruckner Publishing. p. 22. ISBN 978-80-904661-1-1.
14. Weyns, Danny; Omicini, Amdrea; Odell, James (2007). "Environment as a first-class abstraction in multiagent systems". *Autonomous Agents and Multi-Agent Systems*. **14** (1): 5–30. CiteSeerX 10.1.1.154.4480 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.154.4480). doi:10.1007/s10458-006-0012-0 (https://doi.org/10.1007%2Fs10458-006-0012-0). S2CID 13347050 (https://api.semanticscholar.org/CorpusID:13347050).
15. Wooldridge, Michael (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons. p. 366. ISBN 978-0-471-49691-5.
16. Panait, Liviu; Luke, Sean (2005). "Cooperative Multi-Agent Learning: The State of the Art" (http://cs.gmu.edu/~eclab/papers/panait05cooperative.pdf) (PDF). *Autonomous Agents and Multi-Agent Systems*. **11** (3): 387–434. CiteSeerX 10.1.1.307.6671 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.307.6671). doi:10.1007/s10458-005-2631-2 (https://doi.org/10.1007%2Fs10458-005-2631-2). S2CID 19706 (https://api.semanticscholar.org/CorpusID:19706).
17. "The Multi-Agent Systems Lab" (http://mas.cs.umass.edu/). University of Massachusetts Amherst. Retrieved Oct 16, 2009.
18. Albrecht, Stefano; Stone, Peter (2017), "Multiagent Learning: Foundations and Recent Trends. Tutorial", *IJCAI-17 conference* (http://www.cs.utexas.edu/~larg/ijcai17_tutorial/multiagent_learning.pdf) (PDF)
19. Cucker, Felipe; Steve Smale (2007). "The Mathematics of Emergence" (http://ttic.uchicago.edu/~smale/papers/math-of-emergence.pdf) (PDF). *Japanese Journal of Mathematics*. **2**: 197–227. doi:10.1007/s11537-007-0647-x (https://doi.org/10.1007%2Fs11537-007-0647-x). S2CID 2637067 (https://api.semanticscholar.org/CorpusID:2637067). Retrieved 2008-06-09.
20. Shen, Jackie (Jianhong) (2008). "Cucker–Smale Flocking under Hierarchical Leadership" (http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SMJMAP00006800000300 0694000001&idtype=cvips&gifs=yes). *SIAM J. Appl. Math*. **68** (3): 694–719. arXiv:q-bio/0610048 (https://arxiv.org/abs/q-bio/0610048). doi:10.1137/060673254 (https://doi.org/10.1137%2F060673254). S2CID 14655317 (https://api.semanticscholar.org/CorpusID:14655317). Retrieved 2008-06-09.
21. Ahmed, S.; Karsiti, M.N. (2007), "A testbed for control schemes using multi agent nonholonomic robots", *2007 IEEE International Conference on Electro/Information Technology* (http://eprints.utp.edu.my/320/), p. 459, doi:10.1109/EIT.2007.4374547 (https://doi.org/10.1109%2FEIT.2007.4374547), ISBN 978-1-4244-0940-2, S2CID 2734931 (https://api.semanticscholar.org/CorpusID:2734931)
22. "OMG Document – orbos/97-10-05 (Update of Revised MAF Submission)" (https://www.omg.org/cgi-bin/doc?orbos/97-10-05). *www.omg.org*. Retrieved 2019-02-19.

23. Ahmed, Salman; Karsiti, Mohd N.; Agustiawan, Herman (2007). "A development framework for collaborative robots using feedback control". CiteSeerX 10.1.1.98.879 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.98.879).
24. "IEEE IES Technical Committee on Industrial Agents (TC-IA)" (https://tcia.ieee-ies.org/). *tcia.ieee-ies.org*. Retrieved 2019-02-19.
25. Leitão, Paulo; Karnouskos, Stamatis (2015-03-26). *Industrial agents : emerging applications of software agents in industry*. Leitão, Paulo,, Karnouskos, Stamatis. Amsterdam, Netherlands. ISBN 978-0128003411. OCLC 905853947 (https://www.worldcat.org/oclc/905853947).
26. "Film showcase" (http://www.massivesoftware.com/film.html). MASSIVE. Retrieved 28 April 2012.
27. Leitao, Paulo; Karnouskos, Stamatis; Ribeiro, Luis; Lee, Jay; Strasser, Thomas; Colombo, Armando W. (2016). "Smart Agents in Industrial Cyber–Physical Systems" (http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-128744). *Proceedings of the IEEE*. **104** (5): 1086–1101. doi:10.1109/JPROC.2016.2521931 (https://doi.org/10.1109%2FJPROC.2016.2521931). ISSN 0018-9219 (https://www.worldcat.org/issn/0018-9219). S2CID 579475 (https://api.semanticscholar.org/CorpusID:579475).
28. Xiao-Feng Xie, S. Smith, G. Barlow. Schedule-driven coordination for real-time traffic network control (http://www.wiomax.com/team/xie/paper/ICAPS12.pdf). International Conference on Automated Planning and Scheduling (ICAPS), São Paulo, Brazil, 2012: 323–331.
29. Máhr, T. S.; Srour, J.; De Weerdt, M.; Zuidwijk, R. (2010). "Can agents measure up? A comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty". *Transportation Research Part C: Emerging Technologies*. **18**: 99–119. CiteSeerX 10.1.1.153.770 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.153.770). doi:10.1016/j.trc.2009.04.018 (https://doi.org/10.1016%2Fj.trc.2009.04.018).
30. "Generation Expansion Planning Considering Investment Dynamic of Market Participants Using Multi-agent System - IEEE Conference Publication". 2019-12-17. doi:10.1109/SGC.2018.8777904 (https://doi.org/10.1109%2FSGC.2018.8777904). S2CID 199058301 (https://api.semanticscholar.org/CorpusID:199058301).
31. "Distributed Multi-Agent System-Based Load Frequency Control for Multi-Area Power System in Smart Grid - IEEE Journals & Magazine". 2019-12-17. doi:10.1109/TIE.2017.2668983 (https://doi.org/10.1109%2FTIE.2017.2668983). S2CID 31816181 (https://api.semanticscholar.org/CorpusID:31816181).
32. AI can predict your future behaviour with powerful new simulations (https://www.newscientist.com/article/mg24332500-800-ai-can-predict-your-future-behaviour-with-powerful-new-simulations)
33. Hallerbach, S.; Xia, Y.; Eberle, U.; Koester, F. (2018). "Simulation-Based Identification of Critical Scenarios for Cooperative and Automated Vehicles" (https://www.researchgate.net/publication/324194968). *SAE International Journal of Connected and Automated Vehicles*. SAE International. **1** (2): 93. doi:10.4271/2018-01-1066 (https://doi.org/10.4271%2F2018-01-1066).
34. Madrigal, Story by Alexis C. "Inside Waymo's Secret World for Training Self-Driving Cars" (https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/). *The Atlantic*. Retrieved 14 August 2020.
35. Connors, J.; Graham, S.; Mailloux, L. (2018). "Cyber Synthetic Modeling for Vehicle-to-Vehicle Applications". *In International Conference on Cyber Warfare and Security*. Academic Conferences International Limited: 594-XI.

Further reading

- Wooldridge, Michael (2002). *An Introduction to MultiAgent Systems*. John Wiley & Sons. p. 366. ISBN 978-0-471-49691-5.

- Shoham, Yoav; Leyton-Brown, Kevin (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations* (http://www.masfoundations.org). Cambridge University Press. p. 496. ISBN 978-0-521-89943-7.
- Mamadou, Tadiou Koné; Shimazu, A.; Nakajima, T. (August 2000). "The State of the Art in Agent Communication Languages (ACL)" (https://www.researchgate.net/publication/215705246). *Knowledge and Information Systems*. **2** (2): 1–26.
- Hewitt, Carl; Inman, Jeff (Nov–Dec 1991). "DAI Betwixt and Between: From "Intelligent Agents" to Open Systems Science" (https://web.archive.org/web/20170831090048/https://pdfs.semanticscholar.org/7840/bbf6b2fceb014cd3e8eeb2bd81529c7b36b5.pdf) (PDF). *IEEE Transactions on Systems, Man, and Cybernetics*. **21** (6): 1409–1419. doi:10.1109/21.135685 (https://doi.org/10.1109%2F21.135685). S2CID 39080989 (https://api.semanticscholar.org/CorpusID:39080989). Archived from the original (https://pdfs.semanticscholar.org/7840/bbf6b2fceb014cd3e8eeb2bd81529c7b36b5.pdf) (PDF) on 2017-08-31.
- *The Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)*
- Weiss, Gerhard, ed. (1999). *Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence*. MIT Press. ISBN 978-0-262-23203-6.
- Ferber, Jacques (1999). *Multi-Agent Systems: An Introduction to Artificial Intelligence*. Addison-Wesley. ISBN 978-0-201-36048-6.
- Weyns, Danny (2010). *Architecture-Based Design of Multi-Agent Systems*. Springer. ISBN 978-3-642-01063-7.
- Sun, Ron (2006). *Cognition and Multi-Agent Interaction* (http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=0-521-83964-5). Cambridge University Press. ISBN 978-0-521-83964-8.
- Keil, David; Goldin, Dina (2006). Weyns, Danny; Parunak, Van; Michel, Fabien (eds.). *Indirect Interaction in Environments for Multiagent Systems* (https://archive.org/details/environmentsform0000e4ma/page/68). *Environments for Multiagent Systems II*. LNCS 3830. Vol. 3830. Springer. pp. 68–87 (https://archive.org/details/environmentsform0000e4ma/page/68). doi:10.1007/11678809_5 (https://doi.org/10.1007%2F11678809_5). ISBN 978-3-540-32614-4.
- *Whitestein Series in Software Agent Technologies and Autonomic Computing (https://web.archive.org/web/20090114063602/http://www.whitestein.com/series)*, published by Springer Science+Business Media Group
- Salamon, Tomas (2011). *Design of Agent-Based Models : Developing Computer Simulations for a Better Understanding of Social Processes* (http://www.designofagentbasedmodels.info/). Bruckner Publishing. ISBN 978-80-904661-1-1.
- Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (http://aima.cs.berkeley.edu/) (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2
- Fasli, Maria (2007). *Agent-technology for E-commerce*. John Wiley & Sons. p. 480. ISBN 978-0-470-03030-1.
- Cao, Longbing, Gorodetsky, Vladimir, Mitkas, Pericles A. (2009). Agent Mining: The Synergy of Agents and Data Mining (http://www2.computer.org/portal/web/csdl/doi/10.1109/MIS.2009.45), IEEE Intelligent Systems, vol. 24, no. 3, 64-72.

State machine replication

In computer science, **state machine replication** (**SMR**) or **state machine approach** is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas. The approach also provides a framework for understanding and designing replication management protocols.[1]

Contents

Problem definition

Distributed service

In terms of clients and services. Each service comprises one or more servers and exports operations that clients invoke by making requests. Although using a single, centralized server is the simplest way to implement a service, the resulting service can only be as fault tolerant as the processor

executing that server. If this level of fault tolerance is unacceptable, then multiple servers that fail independently can be used. Usually, replicas of a single server are executed on separate processors of a distributed system, and protocols are used to coordinate client interactions with these replicas.

State machine

For the subsequent discussion a *State Machine* will be defined as the following tuple of values [2] (See also Mealy machine and Moore Machine):

- A set of *States*
- A set of *Inputs*
- A set of *Outputs*
- A transition function (Input × State → State)
- An output function (Input × State → Output)
- A distinguished State called Start.

A State Machine begins at the State labeled Start. Each Input received is passed through the transition and output function to produce a new State and an Output. The State is held stable until a new Input is received, while the Output is communicated to the appropriate receiver.

This discussion requires a State Machine to be **deterministic**: multiple copies of the same State Machine begin in the Start state, and receiving the same Inputs in the same order will arrive at the same State having generated the same Outputs.

Typically, systems based on State Machine Replication voluntarily restrict their implementations to use finite-state machines to simplify error recovery.

Fault Tolerance

Determinism is an ideal characteristic for providing fault-tolerance. Intuitively, if multiple copies of a system exist, a fault in one would be noticeable as a difference in the State or Output from the others.

A little deduction shows the minimum number of copies needed for fault-tolerance is three; one which has a fault, and two others to whom we compare State and Output. Two copies are not enough as there is no way to tell which copy is the faulty one.

Further deduction shows a three-copy system can support at most one failure (after which it must repair or replace the faulty copy). If more than one of the copies were to fail, all three States and Outputs might differ, and there would be no way to choose which is the correct one.

In general, a system which supports F failures must have 2F+1 copies (also called replicas).[3] The extra copies are used as evidence to decide which of the copies are correct and which are faulty. Special cases can improve these bounds.[4]

All of this deduction pre-supposes that replicas are experiencing only random independent faults such as memory errors or hard-drive crash. Failures caused by replicas which attempt to lie, deceive, or collude can also be handled by the State Machine Approach, with isolated changes.

Failed replicas are not required to stop; they may continue operating, including generating spurious or incorrect Outputs.

Special Case: Fail-Stop

Theoretically, if a failed replica is guaranteed to stop without generating outputs, only F+1 replicas are required, and clients may accept the first output generated by the system. No existing systems achieve this limit, but it is often used when analyzing systems built on top of a fault-tolerant layer (Since the fault-tolerant layer provides fail-stop semantics to all layers above it).

Special Case: Byzantine Failure

Faults where a replica sends different values in different directions (for instance, the correct Output to some of its fellow replicas and incorrect Outputs to others) are called Byzantine Failures.[5] Byzantine failures may be random, spurious faults, or malicious, intelligent attacks. 2F+1 replicas, with non-cryptographic hashes suffices to survive all non-malicious Byzantine failures (with high probability). Malicious attacks require cryptographic primitives to achieve 2F+1 (using message signatures), or non-cryptographic techniques can be applied but the number of replicas must be increased to 3F+1.[5]

The State Machine Approach

The preceding intuitive discussion implies simple technique for implementing a fault-tolerant service in terms of a State Machine:

1. Place copies of the State Machine on multiple, independent servers.
2. Receive client requests, interpreted as Inputs to the State Machine.
3. Choose an ordering for the Inputs.
4. Execute Inputs in the chosen order on each server.
5. Respond to clients with the Output from the State Machine.
6. Monitor replicas for differences in State or Output.

The remainder of this article develops the details of this technique.

- Step 1 and 2 are outside the scope of this article.
- Step 3 is the critical operation, see Ordering Inputs.
- Step 4 is covered by the State Machine Definition.
- Step 5, see Sending Outputs.
- Step 6, see Auditing and Failure Detection.

The appendix contains discussion on typical extensions used in real-world systems such as Logging, Checkpoints, Reconfiguration, and State Transfer.

Ordering Inputs

The critical step in building a distributed system of State Machines is choosing an order for the Inputs to be processed. Since all non-faulty replicas will arrive at the same State and Output if given the same Inputs, it is imperative that the Inputs are submitted in an equivalent order at each replica. Many solutions have been proposed in the literature.[2][6][7][8][9]

A *Visible Channel* is a communication path between two entities actively participating in the system (such as clients and servers). Example: client to server, server to server

A *Hidden Channel* is a communication path which is not revealed to the system. Example: client to client channels are usually hidden; such as users communicating over a telephone, or a process writing files to disk which are read by another process.

When all communication paths are visible channels and no hidden channels exist, a partial global order (**Causal Order**) may be inferred from the pattern of communications.[8][10] Causal Order may be derived independently by each server. Inputs to the State Machine may be executed in Causal Order, guaranteeing consistent State and Output for all non-faulty replicas.

In open systems, hidden channels are common and a weaker form of ordering must be used. An order of Inputs may be defined using a voting protocol whose results depend only on the visible channels.

The problem of voting for a *single* value by a group of independent entities is called **Consensus**. By extension, a *series* of values may be chosen by a series of consensus instances. This problem becomes difficult when the participants or their communication medium may experience failures.[3]

Inputs may be ordered by their position in the series of consensus instances (**Consensus Order**).[7] Consensus Order may be derived independently by each server. Inputs to the State Machine may be executed in Consensus Order, guaranteeing consistent State and Output for all non-faulty replicas.

> **Optimizing Causal & Consensus Ordering**
> In some cases additional information is available (such as real-time clocks). In these cases, it is possible to achieve more efficient causal or consensus ordering for the Inputs, with a reduced number of messages, fewer message rounds, or smaller message sizes. See references for details [1][4][6][11]
>
> Further optimizations are available when the semantics of State Machine operations are accounted for (such as Read vs Write operations). See references Generalized Paxos.[2][12]

Sending Outputs

Client requests are interpreted as Inputs to the State Machine, and processed into Outputs in the appropriate order. Each replica will generate an Output independently. Non-faulty replicas will always produce the same Output. Before the client response can be sent, faulty Outputs must be filtered out. Typically, a majority of the Replicas will return the same Output, and this Output is sent as the response to the client.

System Failure

> If there is no majority of replicas with the same Output, or if less than a majority of replicas returns an Output, a system failure has occurred. The client response must be the unique Output: FAIL.

Auditing and Failure Detection

The permanent, unplanned compromise of a replica is called a *Failure*. Proof of failure is difficult to obtain, as the replica may simply be slow to respond,[13] or even lie about its status.[5]

Non-faulty replicas will always contain the same State and produce the same Outputs. This invariant enables failure detection by comparing States and Outputs of all replicas. Typically, a replica with State or Output which differs from the majority of replicas is declared faulty.

A common implementation is to pass checksums of the current replica State and recent Outputs among servers. An Audit process at each server restarts the local replica if a deviation is detected.[14] Cryptographic security is not required for checksums.

It is possible that the local server is compromised, or that the Audit process is faulty, and the replica continues to operate incorrectly. This case is handled safely by the Output filter described previously (see Sending Outputs).

Appendix: Extensions

Input Log

In a system with no failures, the Inputs may be discarded after being processed by the State Machine. Realistic deployments must compensate for transient non-failure behaviors of the system such as message loss, network partitions, and slow processors.[14]

One technique is to store the series of Inputs in a log. During times of transient behavior, replicas may request copies of a log entry from another replica in order to fill in missing Inputs.[7]

In general the log is not required to be persistent (it may be held in memory). A persistent log may compensate for extended transient periods, or support additional system features such as Checkpoints, and Reconfiguration.

Checkpoints

If left unchecked a log will grow until it exhausts all available storage resources. For continued operation, it is necessary to forget log entries. In general a log entry may be forgotten when its contents are no longer relevant (for instance if all replicas have processed an Input, the knowledge of the Input is no longer needed).

A common technique to control log size is store a duplicate State (called a **Checkpoint**), then discard any log entries which contributed to the checkpoint. This saves space when the duplicated State is smaller than the size of the log.

Checkpoints may be added to any State Machine by supporting an additional Input called **CHECKPOINT**. Each replica maintains a checkpoint in addition to the current State value. When the log grows large, a replica submits the CHECKPOINT command just like a client request. The system will ensure non-faulty replicas process this command in the same order, after which all log entries before the checkpoint may be discarded.

In a system with checkpoints, requests for log entries occurring before the checkpoint are ignored. Replicas which cannot locate copies of a needed log entry are faulty and must re-join the system (see Reconfiguration).

Reconfiguration

Reconfiguration allows replicas to be added and removed from a system while client requests continue to be processed. Planned maintenance and replica failure are common examples of reconfiguration. Reconfiguration involves Quitting and Joining.

Quitting

When a server detects its State or Output is faulty (see Auditing and Failure Detection), it may selectively exit the system. Likewise, an administrator may manually execute a command to remove a replica for maintenance.

A new Input is added to the State Machine called **QUIT**.[2][6] A replica submits this command to the system just like a client request. All non-faulty replicas remove the quitting replica from the system upon processing this Input. During this time, the replica may ignore all protocol messages. If a majority of non-faulty replicas remain, the quit is successful. If not, there is a System Failure.

Joining

After quitting, a failed server may selectively restart or re-join the system. Likewise, an administrator may add a new replica to the group for additional capacity.

A new Input is added to the State Machine called **JOIN**. A replica submits this command to the system just like a client request. All non-faulty replicas add the joining node to the system upon processing this Input. A new replica must be up-to-date on the system's State before joining (see State Transfer).

State Transfer

When a new replica is made available or an old replica is restarted, it must be brought up to the current State before processing Inputs (see Joining). Logically, this requires applying every Input from the dawn of the system in the appropriate order.

Typical deployments short-circuit the logical flow by performing a State Transfer of the most recent Checkpoint (see Checkpoints). This involves directly copying the State of one replica to another using an out-of-band protocol.

A checkpoint may be large, requiring an extended transfer period. During this time, new Inputs may be added to the log. If this occurs, the new replica must also receive the new Inputs and apply them after the checkpoint is received. Typical deployments add the new replica as an observer to the ordering protocol before beginning the state transfer, allowing the new replica to collect Inputs during this period.

Optimizing State Transfer

Common deployments reduce state transfer times by sending only State components which differ. This requires knowledge of the State Machine internals. Since state transfer is usually an out-of-band protocol, this assumption is not difficult to achieve.

Compression is another feature commonly added to state transfer protocols, reducing the size of the total transfer.

Leader Election (for Paxos)

Paxos[7] is a protocol for solving consensus, and may be used as the protocol for implementing Consensus Order.

Paxos requires a single leader to ensure liveness.[7] That is, one of the replicas must remain leader long enough to achieve consensus on the next operation of the state machine. System behavior is unaffected if the leader changes after every instance, or if the leader changes multiple times per instance. The only requirement is that one replica remains leader long enough to move the system forward.

Conflict Resolution

In general, a leader is necessary only when there is disagreement about which operation to perform,[11] and if those operations conflict in some way (for instance, if they do not commute).[12]

When conflicting operations are proposed, the leader acts as the single authority to set the record straight, defining an order for the operations, allowing the system to make progress.

With Paxos, multiple replicas may believe they are leaders at the same time. This property makes Leader Election for Paxos very simple, and any algorithm which guarantees an 'eventual leader' will work.

Historical background

A number of researchers published articles on the replicated state machine approach in the early 1980s. Anita Borg described an implementation of a fault tolerant operating system based on replicated state machines in a 1983 paper "A message system supporting fault tolerance" (https://www.andrew.cmu.edu/course/15-440/READINGS/borg-1983.pdf). Leslie Lamport also proposed the state machine approach, in his 1984 paper on "Using Time Instead of Timeout In Distributed Systems" (http://research.microsoft.com/en-us/um/people/lamport/pubs/using-time.pdf). Fred Schneider later elaborated the approach in his paper "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial" (http://www.cs.cornell.edu/fbs/publications/SMSurvey.pdf).

Ken Birman developed the virtual synchrony model in a series of papers published between 1985 and 1987. The primary reference to this work is "Exploiting Virtual Synchrony in Distributed Systems" (http://portal.acm.org/citation.cfm?id=37515&dl=ACM&coll=GUIDE), which describes the Isis Toolkit, a system that was used to build the New York and Swiss Stock Exchanges, French Air Traffic Control System, US Navy AEGIS Warship, and other applications.

Recent work by Miguel Castro and Barbara Liskov used the state machine approach in what they call a "Practical Byzantine fault tolerance" (http://www.pmg.lcs.mit.edu/papers/osdi99_html/osdi99.html) architecture that replicates especially sensitive services using a version of Lamport's original state machine approach, but with optimizations that substantially improve performance.

Most recently, there was also created the BFT-SMaRt library,[15] a high-performance Byzantine fault-tolerant state machine replication library developed in Java. This library implements a protocol very similar to PBFT's, plus complementary protocols which offer state transfer and on-the-fly reconfiguration of hosts (i.e., JOIN and LEAVE operations). BFT-SMaRt is the most recent effort to implement state machine replication, still being actively maintained.

Raft, a consensus based algorithm, was developed in 2013.

Motivated by PBFT, Tendermint BFT[16] was introduced for partial asynchronous networks and it is mainly used for Proof of Stake blockchains.

References

1. Schneider, Fred (1990). "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial" (http://ecommons.library.cornell.edu/bitstream/1813/6640/2/86-800.ps) (PS). *ACM Computing Surveys*. **22** (4): 299–319. CiteSeerX 10.1.1.69.1536 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.69.1536). doi:10.1145/98163.98167 (https://doi.org/10.1145%2F98163.98167). S2CID 678818 (https://api.semanticscholar.org/CorpusID:678818).
2. Lamport, Leslie (1978). "The Implementation of Reliable Distributed Multiprocess Systems" (http://research.microsoft.com/users/lamport/pubs/pubs.html#implementation). *Computer Networks*. **2** (2): 95–114. doi:10.1016/0376-5075(78)90045-4 (https://doi.org/10.1016%2F0376-5075%2878%2990045-4). Retrieved 2008-03-13.
3. Lamport, Leslie (2004). "Lower Bounds for Asynchronous Consensus" (http://research.microsoft.com/users/lamport/pubs/pubs.html#lower-bound).
4. Lamport, Leslie; Mike Massa (2004). *Cheap Paxos* (http://research.microsoft.com/users/lamport/pubs/pubs.html#web-dsn-submission). *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2004)*. pp. 307–314. doi:10.1109/DSN.2004.1311900 (https://doi.org/10.1109%2FDSN.2004.1311900). ISBN 978-0-7695-2052-0. S2CID 1325830 (https://api.semanticscholar.org/CorpusID:1325830).
5. Lamport, Leslie; Robert Shostak; Marshall Pease (July 1982). "The Byzantine Generals Problem" (http://research.microsoft.com/users/lamport/pubs/pubs.html#byz). *ACM Transactions on Programming Languages and Systems*. **4** (3): 382–401. CiteSeerX 10.1.1.64.2312 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.64.2312). doi:10.1145/357172.357176 (https://doi.org/10.1145%2F357172.357176). S2CID 55899582 (https://api.semanticscholar.org/CorpusID:55899582). Retrieved 2007-02-02.
6. Lamport, Leslie (1984). "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems" (http://research.microsoft.com/users/lamport/pubs/pubs.html#using-time). *ACM Transactions on Programming Languages and Systems*. **6** (2): 254–280. CiteSeerX 10.1.1.71.1078 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.71.1078). doi:10.1145/2993.2994 (https://doi.org/10.1145%2F2993.2994). S2CID 402171 (https://api.semanticscholar.org/CorpusID:402171). Retrieved 2008-03-13.
7. Lamport, Leslie (May 1998). "The Part-Time Parliament" (http://research.microsoft.com/users/lamport/pubs/pubs.html#lamport-paxos). *ACM Transactions on Computer Systems*. **16** (2): 133–169. doi:10.1145/279227.279229 (https://doi.org/10.1145%2F279227.279229). S2CID 421028 (https://api.semanticscholar.org/CorpusID:421028). Retrieved 2007-02-02.
8. Birman, Kenneth; Thomas Joseph (1987). "Exploiting virtual synchrony in distributed systems". *Proceedings of the 11th ACM Symposium on Operating Systems Principles (SOSP)*. **21** (5): 123. doi:10.1145/37499.37515 (https://doi.org/10.1145%2F37499.37515). hdl:1813/6651 (https://hdl.handle.net/1813%2F6651).
9. Lampson, Butler (1996). "How to Build a Highly Available System Using Consensus" (http://research.microsoft.com/lampson/58-Consensus/Abstract.html). Retrieved 2008-03-13.
10. Lamport, Leslie (July 1978). "Time, Clocks and the Ordering of Events in a Distributed System" (http://research.microsoft.com/users/lamport/pubs/pubs.html#time-clocks). *Communications of the ACM*. **21** (7): 558–565. doi:10.1145/359545.359563 (https://doi.org/10.1145%2F359545.359563). S2CID 215822405 (https://api.semanticscholar.org/CorpusID:215822405). Retrieved 2007-02-02.
11. Lamport, Leslie (2005). "Fast Paxos" (http://research.microsoft.com/users/lamport/pubs/pubs.html#fast-paxos).
12. Lamport, Leslie (2005). "Generalized Consensus and Paxos" (http://research.microsoft.com/users/lamport/pubs/pubs.html#generalized).

13. Fischer, Michael J.; Nancy A. Lynch; Michael S. Paterson (1985). "Impossibility of Distributed Consensus with One Faulty Process" (http://research.microsoft.com/users/lamport/pubs/pubs.html#using-time). *Journal of the Association for Computing Machinery*. **32** (2): 347–382. doi:10.1145/3149.214121 (https://doi.org/10.1145%2F3149.214121). S2CID 207660233 (https://api.semanticscholar.org/CorpusID:207660233). Retrieved 2008-03-13.
14. Chandra, Tushar; Robert Griesemer; Joshua Redstone (2007). *Paxos Made Live – An Engineering Perspective* (http://www.read.seas.harvard.edu/~kohler/class/08w-dsi/chandra07paxos.pdf) (PDF). *PODC '07: 26th ACM Symposium on Principles of Distributed Computing*. pp. 398–407. doi:10.1145/1281100.1281103 (https://doi.org/10.1145%2F1281100.1281103). ISBN 9781595936165. S2CID 207164635 (https://api.semanticscholar.org/CorpusID:207164635).
15. BFT-SMaRt (https://code.google.com/p/bft-smart/). Google Code repository for the BFT-SMaRt replication library.
16. Buchman, E.; Kwon, J.; Milosevic, Z. (2018). "The latest gossip on BFT consensus". arXiv:1807.04938 (https://arxiv.org/abs/1807.04938) [cs.DC (https://arxiv.org/archive/cs.DC)].

External links

- Replicated state machines video on MIT TechTV (https://web.archive.org/web/20120516085919/http://techtv.mit.edu/videos/18986)
- Apache Bookkeeper (https://web.archive.org/web/20150308214635/http://zookeeper.apache.org/bookkeeper/) a replicated log service which can be used to build replicated state machines

Atomic broadcast

In fault-tolerant distributed computing, an **atomic broadcast** or **total order broadcast** is a broadcast where all correct processes in a system of multiple processes receive the same set of messages in the same order; that is, the same sequence of messages.[1][2] The broadcast is termed "atomic" because it either eventually completes correctly at all participants, or all participants abort without side effects. Atomic broadcasts are an important distributed computing primitive.

Contents

Properties

The following properties are usually required from an atomic broadcast protocol:

1. Validity: if a correct participant broadcasts a message, then all correct participants will eventually receive it.
2. Uniform Agreement: if one correct participant receives a message, then all correct participants will eventually receive that message.
3. Uniform Integrity: a message is received by each participant at most once, and only if it was previously broadcast.
4. Uniform Total Order: the messages are totally ordered in the mathematical sense; that is, if any correct participant receives message 1 first and message 2 second, then every other correct participant must receive message 1 before message 2.

Rodrigues and Raynal[3] and Schiper et al.[4] define the integrity and validity properties of atomic broadcast slightly differently.

Note that total order is not equivalent to FIFO order, which requires that if a process sent message 1 before it sent message 2, then all participants must receive message 1 before receiving message 2. It is also not equivalent to "causal order", where if message 2 "depends on" or "occurs after" message 1 then all participants must receive message 2 after receiving message 1. While a strong and useful condition, total order requires only that all participants receive the messages in the same order, but does not place other constraints on that order.[5]

Fault tolerance

Designing an algorithm for atomic broadcasts is relatively easy if it can be assumed that computers will not fail. For example, if there are no failures, atomic broadcast can be achieved simply by having all participants communicate with one "leader" which determines the order of the messages, with the other participants following the leader.

However, real computers are faulty; they fail and recover from failure at unpredictable, possibly inopportune, times. For example, in the follow-the-leader algorithm, what if the leader fails at the wrong time? In such an environment achieving atomic broadcasts is difficult.[1] A number of protocols have been proposed for performing atomic broadcast, under various assumptions about the network, failure models, availability of hardware support for multicast, and so forth.[2]

Equivalent to consensus

In order for the conditions for atomic broadcast to be satisfied, the participants must effectively "agree" on the order of receipt of the messages. Participants recovering from failure, after the other participants have "agreed" an order and started to receive the messages, must be able to learn and comply with the agreed order. Such considerations indicate that in systems with crash failures, atomic broadcast and consensus are equivalent problems.[6]

A value can be proposed by a process for consensus by atomically broadcasting it, and a process can decide a value by selecting the value of the first message which it atomically receives. Thus, consensus can be reduced to atomic broadcast.

Conversely, a group of participants can atomically broadcast messages by achieving consensus regarding the first message to be received, followed by achieving consensus on the next message, and so forth until all the messages have been received. Thus, atomic broadcast reduces to consensus. This was demonstrated more formally and in greater detail by Xavier Défago, et al.[2]

A fundamental result in distributed computing is that achieving consensus in asynchronous systems in which even one crash failure can occur is impossible in the most general case. This was shown in 1985 by Michael J. Fischer, Nancy Lynch, and Mike Paterson, and is sometimes called the FLP result.[7] Since consensus and atomic broadcast are equivalent, FLP applies also to atomic broadcast.[5] The FLP result does not prohibit the implementation of atomic broadcast in practice, but it does require making less stringent assumptions than FLP in some respect, such as about processor and communication timings.

Algorithms

The Chandra-Toueg algorithm[6] is a consensus-based solution to atomic broadcast. Another solution has been put forward by Rodrigues and Raynal.[3]

The Zookeeper Atomic Broadcast (ZAB) protocol is the basic building block for Apache ZooKeeper, a fault-tolerant distributed coordination service which underpins Hadoop and many other important distributed systems.[8][9]

Ken Birman has proposed the virtual synchrony execution model for distributed systems, the idea of which is that all processes observe the same events in the same order. A total ordering of the messages being received, as in atomic broadcast, is one (though not the only) method for attaining virtually synchronous message receipt.

References

1. Kshemkalyani, Ajay; Singhal, Mukesh (2008). *Distributed Computing: Principles, Algorithms, and Systems (Google eBook)* (https://archive.org/details/distributedsyste00ajay_336). Cambridge University Press. pp. 583 (https://archive.org/details/distributedsyste00ajay_336/page/n600)–585. ISBN 9781139470315.
2. Défago, Xavier; Schiper, André; Urbán, Péter (2004). "Total order broadcast and multicast algorithms" (https://infoscience.epfl.ch/record/52563/files/IC_TECH_REPORT_200356.pdf)

(PDF). *ACM Computing Surveys*. **36** (4): 372–421. doi:10.1145/1041680.1041682 (https://doi.org/10.1145%2F1041680.1041682).
3. Rodrigues L, Raynal M.: Atomic Broadcast in Asynchronous Crash-Recovery Distributed Systems [1] (http://portal.acm.org/citation.cfm?id=851790), ICDCS '00: Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS 2000)
4. Ekwall, R.; Schiper, A. (2006). "Solving Atomic Broadcast with Indirect Consensus". *International Conference on Dependable Systems and Networks (DSN'06)* (https://infoscience.epfl.ch/record/83547/files/TechReport_LSR_2006_01.pdf) (PDF). pp. 156–165. doi:10.1109/dsn.2006.65 (https://doi.org/10.1109%2Fdsn.2006.65). ISBN 0-7695-2607-1.
5. Dermot Kelly. "Group Communication" (http://www.cs.nuim.ie/~dkelly/CS402-06/Group%20Communication.htm).
6. Chandra, Tushar Deepak; Toueg, Sam (1996). "Unreliable failure detectors for reliable distributed systems". *Journal of the ACM*. **43** (2): 225–267. doi:10.1145/226643.226647 (https://doi.org/10.1145%2F226643.226647).
7. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson (1985). "Impossibility of Distributed Consensus with One Faulty Process" (https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf) (PDF). *Journal of the ACM*. **32** (2): 374–382. doi:10.1145/3149.214121 (https://doi.org/10.1145%2F3149.214121).
8. Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini, Yahoo! Research (2011). "Zab: High-performance broadcast for primary-backup systems". *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*. pp. 245–256. doi:10.1109/DSN.2011.5958223 (https://doi.org/10.1109%2FDSN.2011.5958223). ISBN 978-1-4244-9233-6. S2CID 206611670 (https://api.semanticscholar.org/CorpusID:206611670).
9. André Medeiros (March 20, 2012). "ZooKeeper's atomic broadcast protocol: Theory and practice" (http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf) (PDF). *Helsinki University of Technology - Laboratory of Theoretical Computer Science*.

Byzantine fault

A **Byzantine fault** (also **Byzantine generals problem**, **interactive consistency**, **source congruency**, **error avalanche**, **Byzantine agreement problem**, and **Byzantine failure**[1]) is a condition of a computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed. The term takes its name from an allegory, the "Byzantine generals problem",[2] developed to describe a situation in which, in order to avoid catastrophic failure of the system, the system's actors must agree on a concerted strategy, but some of these actors are unreliable.

In a Byzantine fault, a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers. It is difficult for the other components to declare it failed and shut it out of the network, because they need to first reach a consensus regarding which component has failed in the first place. **Byzantine fault tolerance** (**BFT**) is the resiliency of a fault-tolerant computer system to such conditions.

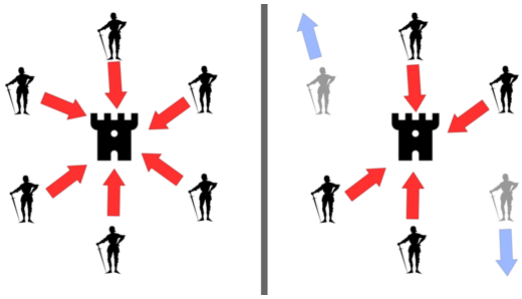
Contents

Analogy

In its simplest form, a number of generals are attacking a fortress and they must decide as a group whether to attack or retreat. Some generals may prefer to attack, while others prefer to retreat. The important thing is that all generals agree on a common decision, for a halfhearted attack by a few generals would become a rout, and would be worse than either a coordinated attack or a coordinated retreat.

The problem is complicated by the presence of treacherous generals who may not only cast a vote for a suboptimal strategy, they may do so selectively. For instance, if nine generals are voting, four of whom support attacking while four others are in favor of retreat, the ninth general may send a vote of retreat to those generals in favor of retreat, and a vote of attack to the rest. Those who received a retreat vote from the ninth general will retreat, while the rest will attack (which may not

go well for the attackers). The problem is complicated further by the generals being physically separated and having to send their votes via messengers who may fail to deliver votes or may forge false votes.



If all generals attack in coordination, the battle is won (left). If two generals falsely declare that they intend to attack, but instead retreat, the battle is lost (right).

Resolution

Byzantine fault tolerance can be achieved if the loyal (non-faulty) generals have a majority agreement on their strategy. There can be a default vote value given to missing messages. For example, missing messages can be given a "null" value. Further, if the agreement is that the null votes are in the majority, a pre-assigned default strategy can be used (e.g. retreat).[3]

The typical mapping of this story onto computer systems is that the computers are the generals and their digital communication system links are the messengers. Although the problem is formulated in the analogy as a decision-making and security problem, in electronics, it cannot be solved by cryptographic digital signatures alone, because failures such as incorrect voltages can propagate through the encryption process. Thus, a component may appear functioning to one component and faulty to another, which prevents forming a consensus as to whether the component is faulty or not.

Characteristics

A Byzantine fault is any fault presenting different symptoms to different observers.[4] A Byzantine failure is the loss of a system service due to a Byzantine fault in systems that require consensus.[5]

The objective of Byzantine fault tolerance is to be able to defend against failures of system components with or without symptoms that prevent other components of the system from reaching an agreement among themselves, where such an agreement is needed for the correct operation of the system.

The remaining operationally correct components of a Byzantine fault tolerant system will be able to continue providing the system's service as originally intended, assuming there are a sufficient number of accurately-operating components to maintain the service.

Byzantine failures are considered the most general and most difficult class of failures among the failure modes. The so-called fail-stop failure mode occupies the simplest end of the spectrum. Whereas fail-stop failure mode simply means that the only way to fail is a node crash, detected by other nodes, Byzantine failures imply no restrictions, which means that the failed node can generate arbitrary data, including data that makes it appear like a functioning node. Thus, Byzantine failures can confuse failure detection systems, which makes fault tolerance difficult. Despite the analogy, a Byzantine failure is not necessarily a security problem involving hostile human interference: it can arise purely from electrical or software faults.

The terms fault and failure are used here according to the standard definitions[6] originally created by a joint committee on "Fundamental Concepts and Terminology" formed by the IEEE Computer Society's Technical Committee on Dependable Computing and Fault-Tolerance and IFIP Working Group 10.4 on Dependable Computing and Fault Tolerance.[7] See also dependability.

Caveat

Byzantine fault tolerance is only concerned with broadcast consistency, that is, the property that when one component broadcasts a single consistent value to other components (i.e., sends the same value to the other components), they all receive exactly the same value, or in the case that the broadcaster is not consistent, the other components agree on a common value. This kind of fault tolerance does not encompass the correctness of the value itself; for example, an adversarial component that deliberately sends an incorrect value, but sends that same value consistently to all components, will not be caught in the Byzantine fault tolerance scheme.

Formal definition

Setting:[8] Given a system of n components, t of which are dishonest, and assuming only point-to-point channel between all the components.

Whenever a component A tries to broadcast a value x, the other components are allowed to discuss with each other and verify the consistency of A's broadcast, and eventually settle on a common value y.

Property: The system is said to resist Byzantine faults if a component A can broadcast a value x, and then:

1. If A is honest, then all honest components agree on the value x.
2. In any case, all honest components agree on the same value y.

Variants: The problem has been studied in the case of both synchronous and asynchronous communications.

The communication graph above is assumed to be the complete graph (i.e. each component can discuss with every other), but the communication graph can be restricted.

It can also be relaxed in a more "realistic" problem where the faulty components do not collude together in an attempt to lure the others into error. It is in this setting that practical algorithms have been devised.

History

The problem of obtaining Byzantine consensus was conceived and formalized by Robert Shostak, who dubbed it the *interactive consistency* problem. This work was done in 1978 in the context of the NASA-sponsored SIFT[9] project in the Computer Science Lab at SRI International. SIFT (for Software Implemented Fault Tolerance) was the brain child of John Wensley, and was based on the idea of using multiple general-purpose computers that would communicate through pairwise messaging in order to reach a consensus, even if some of the computers were faulty.

At the beginning of the project, it was not clear how many computers in total were needed to guarantee that a conspiracy of n faulty computers could not "thwart" the efforts of the correctly-operating ones to reach consensus. Shostak showed that a minimum of $3n+1$ are needed, and devised a two-round $3n+1$ messaging protocol that would work for $n=1$. His colleague Marshall Pease generalized the algorithm for any $n > 0$, proving that $3n+1$ is both necessary and sufficient. These results, together with a later proof by Leslie Lamport of the sufficiency of $3n$ using digital signatures, were published in the seminal paper, *Reaching Agreement in the Presence of Faults*.[10] The authors were awarded the 2005 Edsger W. Dijkstra Prize for this paper.

To make the interactive consistency problem easier to understand, Lamport devised a colorful allegory in which a group of army generals formulate a plan for attacking a city. In its original version, the story cast the generals as commanders of the Albanian army. The name was changed,

eventually settling on "Byzantine", at the suggestion of Jack Goldberg to future-proof any potential offense giving.[11] This formulation of the problem, together with some additional results, were presented by the same authors in their 1982 paper, "The Byzantine Generals Problem".[3]

Examples

Several examples of Byzantine failures that have occurred are given in two equivalent journal papers.[4][5] These and other examples are described on the NASA DASHlink web pages.[12]

Byzantine errors were observed infrequently and at irregular points during endurance testing for the newly constructed *Virginia* class submarines, at least through 2005 (when the issues were publicly reported).[13]

Implementations

One example of BFT in use is Bitcoin, a peer-to-peer digital cash system.[14] The Bitcoin network works in parallel to generate a blockchain with proof-of-work allowing the system to overcome Byzantine failures and reach a coherent global view of the system's state.

Some aircraft systems, such as the Boeing 777 Aircraft Information Management System (via its ARINC 659 SAFEbus network), the Boeing 777 flight control system, and the Boeing 787 flight control systems use Byzantine fault tolerance; because these are real-time systems, their Byzantine fault tolerance solutions must have very low latency. For example, SAFEbus can achieve Byzantine fault tolerance within the order of a microsecond of added latency.[15][16][17] The SpaceX Dragon considers Byzantine fault tolerance in its design.[18]

Byzantine fault tolerance mechanisms use components that repeat an incoming message (or just its signature) to other recipients of that incoming message. All these mechanisms make the assumption that the act of repeating a message blocks the propagation of Byzantine symptoms. For systems that have a high degree of safety or security criticality, these assumptions must be proven to be true to an acceptable level of fault coverage. When providing proof through testing, one difficulty is creating a sufficiently wide range of signals with Byzantine symptoms.[19] Such testing likely will require specialized fault injectors.[20][21]

Solutions

Several early solutions were described by Lamport, Shostak, and Pease in 1982.[3] They began by noting that the Generals' Problem can be reduced to solving a "Commander and Lieutenants" problem where loyal Lieutenants must all act in unison and that their action must correspond to what the Commander ordered in the case that the Commander is loyal:

- One solution considers scenarios in which messages may be forged, but which will be *Byzantine-fault-tolerant* as long as the number of disloyal generals is less than one third of the generals. The impossibility of dealing with one-third or more traitors ultimately reduces to proving that the one Commander and two Lieutenants problem cannot be solved, if the Commander is traitorous. To see this, suppose we have a traitorous Commander A, and two Lieutenants, B and C: when A tells B to attack and C to retreat, and B and C send messages to each other, forwarding A's message, neither B nor C can figure out who is the traitor, since it is not necessarily A—the other Lieutenant could have forged the message purportedly from A. It can be shown that if n is the number of generals in total, and t is the number of traitors in that n, then there are solutions to the problem only when $n > 3t$ and the communication is synchronous (bounded delay).[22]

- A second solution requires unforgeable message signatures. For security-critical systems, digital signatures (in modern computer systems, this may be achieved in practice using public-key cryptography) can provide Byzantine fault tolerance in the presence of an arbitrary number of traitorous generals. However, for safety-critical systems (where "security" addresses intelligent threats while "safety" addresses the inherent dangers of an activity or mission), simple error detecting codes, such as CRCs, provide weaker but often sufficient coverage at a much lower cost. This is true for both Byzantine and non-Byzantine faults. Furthermore, sometimes security measures weaken safety and vice versa. Thus, cryptographic digital signature methods are not a good choice for safety-critical systems, unless there is also a specific security threat as well.[23] While error detecting codes, such as CRCs, are better than cryptographic techniques, neither provide adequate coverage for active electronics in safety-critical systems. This is illustrated by the *Schrödinger CRC* scenario where a CRC-protected message with a single Byzantine faulty bit presents different data to different observers and each observer sees a valid CRC.[4][5]
- Also presented is a variation on the first two solutions allowing Byzantine-fault-tolerant behavior in some situations where not all generals can communicate directly with each other.

Several system architectures were designed c. 1980 that implemented Byzantine fault tolerance. These include: Draper's FTMP,[24] Honeywell's MMFCS,[25] and SRI's SIFT.[9]

Advanced solutions

In 1999, Miguel Castro and Barbara Liskov introduced the "Practical Byzantine Fault Tolerance" (PBFT) algorithm,[26] which provides high-performance Byzantine state machine replication, processing thousands of requests per second with sub-millisecond increases in latency.

After PBFT, several BFT protocols were introduced to improve its robustness and performance. For instance, Q/U,[27] HQ,[28] Zyzzyva,[29] and ABsTRACTs,[30] addressed the performance and cost issues; whereas other protocols, like Aardvark[31] and RBFT,[32] addressed its robustness issues. Furthermore, Adapt[33] tried to make use of existing BFT protocols, through switching between them in an adaptive way, to improve system robustness and performance as the underlying conditions change. Furthermore, BFT protocols were introduced that leverage trusted components to reduce the number of replicas, e.g., A2M-PBFT-EA[34] and MinBFT.[35]

Motivated by PBFT, Tendermint BFT[36] was introduced for partial asynchronous networks and it is mainly used for Proof of Stake blockchains.

While tolerating Byzantine faults is an ideal property, it requires at least four independent administrative domains. To mitigate the limitation, Scalar DL was introduced to realize "Practical Byzantine Fault Detection"[37] algorithm, which detects Byzantine faults with only two administrative domains in a transactional database system.

See also

- Atomic commit
- Brooks–Iyengar algorithm
- List of terms relating to algorithms and data structures
- Byzantine Paxos
- Quantum Byzantine agreement
- Two Generals' Problem

References

1. Kirrmann, Hubert (n.d.). "Fault Tolerant Computing in Industrial Automation" (https://web.archive.org/web/20140326192930/http://lamspeople.epfl.ch/kirrmann/Pubs/FaultTolerance/Fault_Tolerance_Tutorial_HK.pdf#page=94#page=94) (PDF). Switzerland: ABB Research Center. p. 94. Archived from the original (http://lamspeople.epfl.ch/kirrmann/Pubs/FaultTolerance/Fault_Tolerance_Tutorial_HK.pdf#page=94) (PDF) on 2014-03-26. Retrieved 2015-03-02.
2. Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" (https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf) (PDF). *ACM Transactions on Programming Languages and Systems*. **4** (3): 382–401. CiteSeerX 10.1.1.64.2312 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.64.2312). doi:10.1145/357172.357176 (https://doi.org/10.1145%2F357172.357176). Archived (https://web.archive.org/web/20180613015025/https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/?from=http%3A%2F%2Fresearch.microsoft.com%2Fen-us%2Fum%2Fpeople%2Flamport%2Fpubs%2Fbyz.pdf) (PDF) from the original on 13 June 2018.
3. Lamport, L.; Shostak, R.; Pease, M. (1982). "The Byzantine Generals Problem" (https://web.archive.org/web/20170207104645/http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf) (PDF). *ACM Transactions on Programming Languages and Systems*. **4** (3): 387–389. CiteSeerX 10.1.1.64.2312 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.64.2312). doi:10.1145/357172.357176 (https://doi.org/10.1145%2F357172.357176). Archived from the original (http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf) (PDF) on 7 February 2017.
4. Driscoll, K.; Hall, B.; Paulitsch, M.; Zumsteg, P.; Sivencrona, H. (2004). "The Real Byzantine Generals". *The 23rd Digital Avionics Systems Conference (IEEE Cat. No.04CH37576)*. pp. 6.D.4–61–11. doi:10.1109/DASC.2004.1390734 (https://doi.org/10.1109%2FDASC.2004.1390734). ISBN 978-0-7803-8539-9. S2CID 15549497 (https://api.semanticscholar.org/CorpusID:15549497).
5. Driscoll, Kevin; Hall, Brendan; Sivencrona, Håkan; Zumsteg, Phil (2003). "Byzantine Fault Tolerance, from Theory to Reality". *Computer Safety, Reliability, and Security*. Lecture Notes in Computer Science. Vol. 2788. pp. 235–248. doi:10.1007/978-3-540-39878-3_19 (https://doi.org/10.1007%2F978-3-540-39878-3_19). ISBN 978-3-540-20126-7. ISSN 0302-9743 (https://www.worldcat.org/issn/0302-9743). S2CID 12690337 (https://api.semanticscholar.org/CorpusID:12690337).
6. Avizienis, A.; Laprie, J.-C.; Randell, Brian; Landwehr, C. (2004). "Basic concepts and taxonomy of dependable and secure computing". *IEEE Transactions on Dependable and Secure Computing*. **1** (1): 11–33. doi:10.1109/TDSC.2004.2 (https://doi.org/10.1109%2FTDSC.2004.2). hdl:1903/6459 (https://hdl.handle.net/1903%2F6459). ISSN 1545-5971 (https://www.worldcat.org/issn/1545-5971). S2CID 215753451 (https://api.semanticscholar.org/CorpusID:215753451).
7. "Dependable Computing and Fault Tolerance" (http://www.dependability.org). Archived (https://web.archive.org/web/20150402141319/http://www.dependability.org/) from the original on 2015-04-02. Retrieved 2015-03-02.
8. Matthias Fitzi (2002). "Generalized Communication and Security Models in Byzantine Agreement" (https://www.crypto.ethz.ch/publications/files/Fitzi03.pdf) (PDF). *ETH Zurich*.
9. "SIFT: design and analysis of a fault-tolerant computer for aircraft control". *Microelectronics Reliability*. **19** (3): 190. 1979. doi:10.1016/0026-2714(79)90211-7 (https://doi.org/10.1016%2F0026-2714%2879%2990211-7). ISSN 0026-2714 (https://www.worldcat.org/issn/0026-2714).
10. Pease, Marshall; Shostak, Robert; Lamport, Leslie (April 1980). "Reaching Agreement in the Presence of Faults". *Journal of the Association for Computing Machinery*. **27** (2): 228–234. CiteSeerX 10.1.1.68.4044 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.4044). doi:10.1145/322186.322188 (https://doi.org/10.1145%2F322186.322188). S2CID 6429068 (https://api.semanticscholar.org/CorpusID:6429068).

11. Lamport, Leslie (2016-12-19). "The Byzantine Generals Problem" (https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/). *ACM Transactions on Programming Languages and Systems*. SRI International. Retrieved 18 March 2019.
12. Driscoll, Kevin (2012-12-11). "Real System Failures" (https://c3.nasa.gov/dashlink/resources/624/). *DASHlink*. NASA. Archived (https://web.archive.org/web/20150402190610/https://c3.nasa.gov/dashlink/resources/624/) from the original on 2015-04-02. Retrieved 2015-03-02.
13. Walter, C.; Ellis, P.; LaValley, B. (2005). "The Reliable Platform Service: A Property-Based Fault Tolerant Service Architecture". *Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE'05)*. pp. 34–43. doi:10.1109/HASE.2005.23 (https://doi.org/10.1109%2FHASE.2005.23). ISBN 978-0-7695-2377-4. S2CID 21468069 (https://api.semanticscholar.org/CorpusID:21468069).
14. "Bitcoin - Open source P2P money" (https://bitcoin.org/en/). *bitcoin.org*. Retrieved 2019-08-18.
15. M., Paulitsch; Driscoll, K. (9 January 2015). "Chapter 48:SAFEbus" (https://books.google.com/books?id=ppzNBQAAQBAJ). In Zurawski, Richard (ed.). *Industrial Communication Technology Handbook, Second Edition*. CRC Press. pp. 48–1–48–26. ISBN 978-1-4822-0733-0.
16. Thomas A. Henzinger; Christoph M. Kirsch (26 September 2001). *Embedded Software: First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October 8-10, 2001. Proceedings* (http://www.csl.sri.com/papers/emsoft01/emsoft01.pdf) (PDF). Springer Science & Business Media. pp. 307–. ISBN 978-3-540-42673-8. Archived (https://web.archive.org/web/20150922114036/http://www.csl.sri.com/papers/emsoft01/emsoft01.pdf) (PDF) from the original on 2015-09-22. Retrieved 2015-03-05.
17. Yeh, Y.C. (2001). "Safety critical avionics for the 777 primary flight controls system". *20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219)*. Vol. 1. pp. 1C2/1–1C2/11. doi:10.1109/DASC.2001.963311 (https://doi.org/10.1109%2FDASC.2001.963311). ISBN 978-0-7803-7034-0. S2CID 61489128 (https://api.semanticscholar.org/CorpusID:61489128).
18. "ELC: SpaceX lessons learned [LWN.net]" (https://lwn.net/Articles/540368/). Archived (https://web.archive.org/web/20160805064218/http://lwn.net/Articles/540368/) from the original on 2016-08-05. Retrieved 2016-07-21.
19. Nanya, T.; Goosen, H.A. (1989). "The Byzantine hardware fault model". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. **8** (11): 1226–1231. doi:10.1109/43.41508 (https://doi.org/10.1109%2F43.41508). ISSN 0278-0070 (https://www.worldcat.org/issn/0278-0070).
20. Martins, Rolando; Gandhi, Rajeev; Narasimhan, Priya; Pertet, Soila; Casimiro, António; Kreutz, Diego; Veríssimo, Paulo (2013). "Experiences with Fault-Injection in a Byzantine Fault-Tolerant Protocol". *Middleware 2013*. Lecture Notes in Computer Science. Vol. 8275. pp. 41–61. doi:10.1007/978-3-642-45065-5_3 (https://doi.org/10.1007%2F978-3-642-45065-5_3). ISBN 978-3-642-45064-8. ISSN 0302-9743 (https://www.worldcat.org/issn/0302-9743).
21. US patent 7475318 (https://worldwide.espacenet.com/textdoc?DB=EPODOC&IDX=US7475318), Kevin R. Driscoll, "Method for testing the sensitive input range of Byzantine filters", issued 2009-01-06, assigned to Honeywell International Inc.
22. Feldman, P.; Micali, S. (1997). "An optimal probabilistic protocol for synchronous Byzantine agreement" (http://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Distributed%20Computation/An%20Optimal%20Probabilistic%20Algorithm%20for%20Byzantine%20Agreement.pdf) (PDF). *SIAM J. Comput.* **26** (4): 873–933. doi:10.1137/s0097539790187084 (https://doi.org/10.1137%2Fs0097539790187084). Archived (https://web.archive.org/web/20160305012505/http://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Distributed%20Computation/An%20Optimal%20Probabilistic%20Algorithm%20for%20Byzantine%20Agreement.pdf) (PDF) from the original on 2016-03-05. Retrieved 2012-06-14.

23. Paulitsch, M.; Morris, J.; Hall, B.; Driscoll, K.; Latronico, E.; Koopman, P. (2005). "Coverage and the Use of Cyclic Redundancy Codes in Ultra-Dependable Systems". *2005 International Conference on Dependable Systems and Networks (DSN'05)* (https://figshare.com/articles/journal_contribution/Coverage_and_the_Use_of_Cyclic_Redundancy_Codes_in_Ultra-Dependable_Systems/6621788). pp. 346–355. doi:10.1109/DSN.2005.31 (https://doi.org/10.1109%2FDSN.2005.31). ISBN 978-0-7695-2282-1. S2CID 14096385 (https://api.semanticscholar.org/CorpusID:14096385).
24. Hopkins, Albert L.; Lala, Jaynarayan H.; Smith, T. Basil (1987). "The Evolution of Fault Tolerant Computing at the Charles Stark Draper Laboratory, 1955–85". *The Evolution of Fault-Tolerant Computing*. Dependable Computing and Fault-Tolerant Systems. Vol. 1. pp. 121–140. doi:10.1007/978-3-7091-8871-2_6 (https://doi.org/10.1007%2F978-3-7091-8871-2_6). ISBN 978-3-7091-8873-6. ISSN 0932-5581 (https://www.worldcat.org/issn/0932-5581).
25. Driscoll, Kevin; Papadopoulos, Gregory; Nelson, Scott; Hartmann, Gary; Ramohalli, Gautham (1984), *Multi-Microprocessor Flight Control System* (Technical Report), Wright-Patterson Air Force Base, OH: AFWAL/FIGL U.S. Air Force Systems Command, AFWAL-TR-84-3076
26. Castro, M.; Liskov, B. (2002). "Practical Byzantine Fault Tolerance and Proactive Recovery". *ACM Transactions on Computer Systems*. Association for Computing Machinery. **20** (4): 398–461. CiteSeerX 10.1.1.127.6130 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.6130). doi:10.1145/571637.571640 (https://doi.org/10.1145%2F571637.571640). S2CID 18793794 (https://api.semanticscholar.org/CorpusID:18793794).
27. Abd-El-Malek, M.; Ganger, G.; Goodson, G.; Reiter, M.; Wylie, J. (2005). "Fault-scalable Byzantine Fault-Tolerant Services". *ACM SIGOPS Operating Systems Review*. Association for Computing Machinery. **39** (5): 59. doi:10.1145/1095809.1095817 (https://doi.org/10.1145%2F1095809.1095817).
28. Cowling, James; Myers, Daniel; Liskov, Barbara; Rodrigues, Rodrigo; Shrira, Liuba (2006). *HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance* (http://portal.acm.org/citation.cfm?id=1298455.1298473). Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation. pp. 177–190. ISBN 1-931971-47-1.
29. Kotla, Ramakrishna; Alvisi, Lorenzo; Dahlin, Mike; Clement, Allen; Wong, Edmund (December 2009). "Zyzzyva: Speculative Byzantine Fault Tolerance". *ACM Transactions on Computer Systems*. Association for Computing Machinery. **27** (4): 1–39. doi:10.1145/1658357.1658358 (https://doi.org/10.1145%2F1658357.1658358).
30. Guerraoui, Rachid; Kneževic, Nikola; Vukolic, Marko; Quéma, Vivien (2010). *The Next 700 BFT Protocols* (http://infoscience.epfl.ch/record/144158). Proceedings of the 5th European conference on Computer systems. EuroSys. Archived (https://web.archive.org/web/20111002225957/http://infoscience.epfl.ch/record/144158) from the original on 2011-10-02. Retrieved 2011-10-04.
31. Clement, A.; Wong, E.; Alvisi, L.; Dahlin, M.; Marchetti, M. (April 22–24, 2009). *Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults* (http://www.usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf) (PDF). Symposium on Networked Systems Design and Implementation. USENIX. Archived (https://web.archive.org/web/20101225155052/https://www.usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf) (PDF) from the original on 2010-12-25. Retrieved 2010-02-17.
32. Aublin, P.-L.; Ben Mokhtar, S.; Quéma, V. (July 8–11, 2013). *RBFT: Redundant Byzantine Fault Tolerance* (https://web.archive.org/web/20130805115252/http://www.temple.edu/cis/icdcs2013/program.html). 33rd IEEE International Conference on Distributed Computing Systems. International Conference on Distributed Computing Systems. Archived from the original (http://www.temple.edu/cis/icdcs2013/program.html) on August 5, 2013.
33. Bahsoun, J. P.; Guerraoui, R.; Shoker, A. (2015-05-01). "Making BFT Protocols Really Adaptive" (http://repositorio.inesctec.pt/handle/123456789/4107). *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*: 904–913. doi:10.1109/IPDPS.2015.21 (https://doi.org/10.1109%2FIPDPS.2015.21). ISBN 978-1-4799-8649-1. S2CID 16310807 (https://api.semanticscholar.org/CorpusID:16310807).

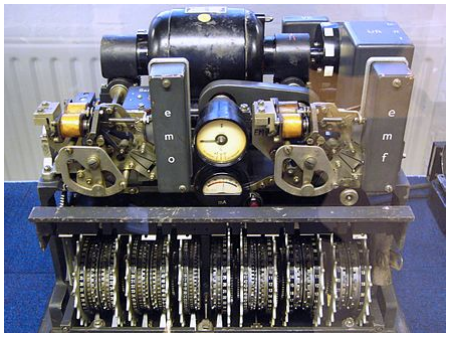
34. Chun, Byung-Gon; Maniatis, Petros; Shenker, Scott; Kubiatowicz, John (2007-01-01). "Attested Append-only Memory: Making Adversaries Stick to Their Word". *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*. SOSP '07. New York, NY, USA: ACM: 189–204. doi:10.1145/1294261.1294280 (https://doi.org/10.1145%2F1294261.1294280). ISBN 9781595935915. S2CID 6685352 (https://api.semanticscholar.org/CorpusID:6685352).
35. Veronese, G. S.; Correia, M.; Bessani, A. N.; Lung, L. C.; Verissimo, P. (2013-01-01). "Efficient Byzantine Fault-Tolerance". *IEEE Transactions on Computers*. **62** (1): 16–30. CiteSeerX 10.1.1.408.9972 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.408.9972). doi:10.1109/TC.2011.221 (https://doi.org/10.1109%2FTC.2011.221). ISSN 0018-9340 (https://www.worldcat.org/issn/0018-9340). S2CID 8157723 (https://api.semanticscholar.org/CorpusID:8157723).
36. Buchman, E.; Kwon, J.; Milosevic, Z. (2018). "The latest gossip on BFT consensus". arXiv:1807.04938 (https://arxiv.org/abs/1807.04938) [cs.DC (https://arxiv.org/archive/cs.DC)].
37. Yamada, Hiroyuki; Nemoto, Jun (2022-03-01). "Scalar DL: scalable and practical byzantine fault detection for transactional database systems" (https://doi.org/10.14778/3523210.3523212). *Proceedings of the VLDB Endowment*. **15** (7): 1324–1336. doi:10.14778/3523210.3523212 (https://doi.org/10.14778%2F3523210.3523212). ISSN 2150-8097 (https://www.worldcat.org/issn/2150-8097).

External links

- Byzantine Fault Tolerance in the RKBExplorer (https://web.archive.org/web/20080828060019/http://www.rkbexplorer.com/explorer/#display=mechanism%2D{http://resex.rkbexplorer.com/id/resilience-mechanism-65b5cef4})

Cryptography

Cryptography, or **cryptology** (from Ancient Greek: κρυπτός, romanized: *kryptós* "hidden, secret"; and γράφειν *graphein*, "to write", or -λογία *-logia*, "study", respectively[1]), is the practice and study of techniques for secure communication in the presence of adversarial behavior.[2] More generally, cryptography is about constructing and analyzing protocols that prevent third parties or the public from reading private messages.[3] Modern cryptography exists at the intersection of the disciplines of mathematics, computer science, information security, electrical engineering, digital signal processing, physics, and others.[4][5] Core concepts related to information security (data confidentiality, data integrity, authentication, and non-repudiation) are also central to cryptography.[6] Practical applications of cryptography include electronic commerce, chip-based payment cards, digital currencies, computer passwords, and military communications.[5]



Lorenz cipher machine, used in World War II to encrypt communications of the German High Command

Cryptography prior to the modern age was effectively synonymous with encryption, converting readable information (plaintext) to unintelligible nonsense text (ciphertext), which can only be read by reversing the process (decryption). The sender of an encrypted (coded) message shares the decryption (decoding) technique only with intended recipients to preclude access from adversaries. The cryptography literature often uses the names "Alice" (or "A") for the sender, "Bob" (or "B") for the intended recipient, and "Eve" (or "E") for the evesdropper adversary.[7] Since the development of rotor cipher machines in World War I and the advent of computers in World War II, cryptography methods have become increasingly complex and their applications more varied.

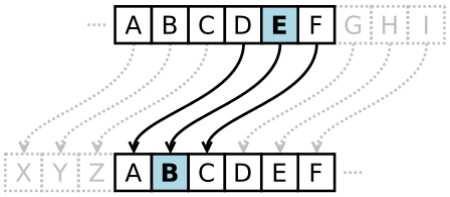
Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are therefore termed "computationally secure"; theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually reevaluated, and if necessary, adapted. Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power, such as the one-time pad, are much more difficult to use in practice than the best theoretically breakable, but computationally secure, schemes.

The growth of cryptographic technology has raised a number of legal issues in the Information Age. Cryptography's potential for use as a tool for espionage and sedition has led many governments to classify it as a weapon and to limit or even prohibit its use and export.[8] In some jurisdictions where the use of cryptography is legal, laws permit investigators to compel the disclosure of encryption keys for documents relevant to an investigation.[9][10] Cryptography also plays a major role in digital rights management and copyright infringement disputes in regard to digital media.[11]

Contents

Terminology

The first use of the term "cryptograph" (as opposed to "cryptogram") dates back to the 19th century—originating from "The Gold-Bug," a story by Edgar Allan Poe.[12][13]



Alphabet shift ciphers are believed to have been used by Julius Caesar over 2,000 years ago.[7] This is an example with $k = 3$. In other words, the letters in the alphabet are shifted three in one direction to encrypt and three in the other direction to decrypt.

Until modern times, cryptography referred almost exclusively to "encryption", which is the process of converting ordinary information (called plaintext) into an unintelligible form (called ciphertext).[14] Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A cipher (or cypher) is a pair of algorithms that carry out the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and, in each instance, by a "key". The key is a secret (ideally known only to the communicants), usually a string of characters (ideally short so it can be remembered by the user), which is

needed to decrypt the ciphertext. In formal mathematical terms, a "cryptosystem" is the ordered list of elements of finite possible plaintexts, finite possible cyphertexts, finite possible keys, and the encryption and decryption algorithms that correspond to each key. Keys are important both formally and in actual practice, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.

There are two main types of cryptosystems: symmetric and asymmetric. In symmetric systems, the only ones known until the 1970s, the same secret key encrypts and decrypts a message. Data manipulation in symmetric systems is significantly faster than in asymmetric systems. Asymmetric systems use a "public key" to encrypt a message and a related "private key" to decrypt it. The advantage of asymmetric systems is that the public key can be freely published, allowing parties to establish secure communication without having a shared secret key. In practice, asymmetric systems are used to first exchange a secret key, and then secure communication proceeds via a more efficient symmetric system using that key.[15] Examples of asymmetric systems include Diffie–Hellman key exchange, RSA (Rivest–Shamir–Adleman), ECC (Elliptic Curve Cryptography), and Post-quantum cryptography. Secure symmetric algorithms include the commonly used AES (Advanced Encryption Standard) which replaced the older DES (Data Encryption Standard).[16] Insecure symmetric algorithms include children's language tangling schemes such as Pig Latin or other cant, and all historical cryptographic schemes, however seriously intended, prior to the invention of the one-time pad early in the 20th century.

In colloquial use, the term "code" is often used to mean any method of encryption or concealment of meaning. However, in cryptography, code has a more specific meaning: the replacement of a unit of plaintext (i.e., a meaningful word or phrase) with a code word (for example, "wallaby" replaces "attack at dawn"). A cypher, in contrast, is a scheme for changing or substituting an element below such a level (a letter, a syllable, or a pair of letters, etc.) in order to produce a cyphertext.

Cryptanalysis is the term used for the study of methods for obtaining the meaning of encrypted information without access to the key normally required to do so; i.e., it is the study of how to "crack" encryption algorithms or their implementations.

Some use the terms "cryptography" and "cryptology" interchangeably in English,[17] while others (including US military practice generally) use "cryptography" to refer specifically to the use and practice of cryptographic techniques and "cryptology" to refer to the combined study of cryptography and cryptanalysis.[18][19] English is more flexible than several other languages in which "cryptology" (done by cryptologists) is always used in the second sense above. RFC 2828 (https://datatracker.ietf.org/doc/html/rfc2828) advises that steganography is sometimes included in cryptology.[20]

The study of characteristics of languages that have some application in cryptography or cryptology (e.g. frequency data, letter combinations, universal patterns, etc.) is called cryptolinguistics.

History

Before the modern era, cryptography focused on message confidentiality (i.e., encryption)—conversion of messages from a comprehensible form into an incomprehensible one and back again at the other end, rendering it unreadable by interceptors or eavesdroppers without secret knowledge (namely the key needed for decryption of that message). Encryption attempted to ensure secrecy in communications, such as those of spies, military leaders, and diplomats. In

recent decades, the field has expanded beyond confidentiality concerns to include techniques for message integrity checking, sender/receiver identity authentication, digital signatures, interactive proofs and secure computation, among others.

Classic cryptography

Reconstructed ancient Greek *scytale*, an early cipher device

The main classical cipher types are transposition ciphers, which rearrange the order of letters in a message (e.g., 'hello world' becomes 'ehlol owrdl' in a trivially simple rearrangement scheme), and substitution ciphers, which systematically replace letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet).[21] Simple versions of either have never offered much confidentiality from enterprising opponents. An early substitution cipher was the Caesar cipher, in which each letter in the plaintext was replaced by a letter some fixed number of positions further down the alphabet. Suetonius reports that Julius Caesar used it with a shift of three to communicate with his generals. Atbash is an example of an early Hebrew cipher. The earliest known use of cryptography is some carved ciphertext on stone in Egypt (ca 1900 BCE), but this may have been done for the amusement of literate observers rather than as a way of concealing information.

The Greeks of Classical times are said to have known of ciphers (e.g., the scytale transposition cipher claimed to have been used by the Spartan military).[22] Steganography (i.e., hiding even the existence of a message so as to keep it confidential) was also first developed in ancient times. An early example, from Herodotus, was a message tattooed on a slave's shaved head and concealed under the regrown hair.[14] More modern examples of steganography include the use of invisible ink, microdots, and digital watermarks to conceal information.

In India, the 2000-year-old Kamasutra of Vātsyāyana speaks of two different kinds of ciphers called Kautiliyam and Mulavediya. In the Kautiliyam, the cipher letter substitutions are based on phonetic relations, such as vowels becoming consonants. In the Mulavediya, the cipher alphabet consists of pairing letters and using the reciprocal ones.[14]

In Sassanid Persia, there were two secret scripts, according to the Muslim author Ibn al-Nadim: the *šāh-dabīrīya* (literally "King's script") which was used for official correspondence, and the *rāz-saharīya* which was used to communicate secret messages with other countries.[23]

David Kahn notes in *The Codebreakers* that modern cryptology originated among the Arabs, the first people to systematically document cryptanalytic methods.[24] Al-Khalil (717–786) wrote the *Book of Cryptographic Messages*, which contains the first use of permutations and combinations to list all possible Arabic words with and without vowels.[25]

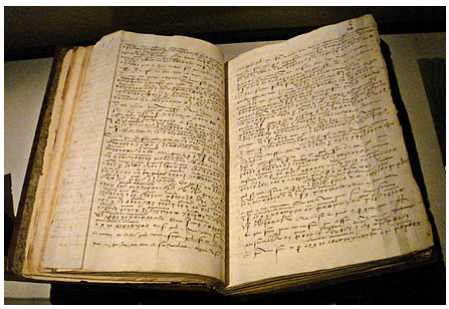
Ciphertexts produced by a classical cipher (and some modern ciphers) will reveal statistical information about the plaintext, and that information can often be used to break the cipher. After the discovery of frequency analysis, perhaps by the Arab mathematician and polymath Al-Kindi (also known as *Alkindus*) in the 9th century,[26] nearly all such ciphers could be broken by an informed attacker. Such classical ciphers still enjoy popularity today, though mostly as puzzles (see cryptogram). Al-Kindi wrote a book on cryptography entitled *Risalah fi Istikhraj al-Mu'amma* (*Manuscript for the Deciphering Cryptographic Messages*), which described the first known use of frequency analysis cryptanalysis techniques.[26][27]

Language letter frequencies may offer little help for some extended historical encryption techniques such as homophonic cipher that tend to flatten the frequency distribution. For those ciphers, language letter group (or n-gram) frequencies may provide an attack.

Essentially all ciphers remained vulnerable to cryptanalysis using the frequency analysis technique until the development of the polyalphabetic cipher, most clearly by Leon Battista Alberti around the year 1467, though there is some indication that it was already known to Al-Kindi.[27] Alberti's innovation was to use different ciphers (i.e., substitution alphabets) for various parts of a message (perhaps for each successive plaintext letter at the limit). He also invented what was probably the first automatic cipher device, a wheel which implemented a partial realization of his invention. In the Vigenère cipher, a polyalphabetic cipher, encryption uses a *key word*, which controls letter substitution depending on which letter of the key word is used. In the mid-19th century Charles Babbage showed that the Vigenère cipher was vulnerable to Kasiski examination, but this was first published about ten years later by Friedrich Kasiski.[28]

First page of a book by Al-Kindi which discusses encryption of messages

16th-century book-shaped French cipher machine, with arms of Henri II of France



Enciphered letter from Gabriel de Luetz d'Aramon, French Ambassador to the Ottoman Empire, after 1546, with partial decipherment

Although frequency analysis can be a powerful and general technique against many ciphers, encryption has still often been effective in practice, as many a would-be cryptanalyst was unaware of the technique. Breaking a message without using frequency analysis essentially required knowledge of the cipher used and perhaps of the key involved, thus making espionage, bribery, burglary, defection, etc., more attractive approaches to the cryptanalytically uninformed. It was finally explicitly recognized in the 19th century that secrecy of a cipher's algorithm is not a sensible nor practical safeguard of message security; in fact, it was further realized that any adequate cryptographic scheme (including ciphers) should remain secure even if the adversary fully understands the cipher algorithm itself. Security of the key used should alone be sufficient for a good cipher to maintain confidentiality under an attack. This fundamental principle was first explicitly stated in 1883 by Auguste Kerckhoffs and is generally called Kerckhoffs's Principle; alternatively and more bluntly, it was restated by Claude Shannon, the inventor of information theory and the fundamentals of theoretical cryptography, as *Shannon's Maxim* —'the enemy knows the system'.

Different physical devices and aids have been used to assist with ciphers. One of the earliest may have been the scytale of ancient Greece, a rod supposedly used by the Spartans as an aid for a transposition cipher. In medieval times, other aids were invented such as the cipher grille, which was also used for a kind of steganography. With the invention of polyalphabetic ciphers came more sophisticated aids such as Alberti's own cipher disk, Johannes Trithemius' tabula recta scheme, and Thomas Jefferson's wheel cypher (not publicly known, and reinvented independently by Bazeries around 1900). Many mechanical encryption/decryption devices were invented early in the 20th century, and several patented, among them rotor machines—famously including the Enigma

machine used by the German government and military from the late 1920s and during World War II.[29] The ciphers implemented by better quality examples of these machine designs brought about a substantial increase in cryptanalytic difficulty after WWI.[30]

Early computer-era cryptography

Cryptanalysis of the new mechanical ciphering devices proved to be both difficult and laborious. In the United Kingdom, cryptanalytic efforts at Bletchley Park during WWII spurred the development of more efficient means for carrying out repetitious tasks, such as military code breaking (decryption). This culminated in the development of the Colossus, the world's first fully electronic, digital, programmable computer, which assisted in the decryption of ciphers generated by the German Army's Lorenz SZ40/42 machine.

Extensive open academic research into cryptography is relatively recent, beginning in the mid-1970s. In the early 1970s IBM personnel designed the Data Encryption Standard (DES) algorithm that became the first federal government cryptography standard in the United States.[31] In 1976 Whitfield Diffie and Martin Hellman published the Diffie–Hellman key exchange algorithm.[32] In 1977 the RSA algorithm was published in Martin Gardner's *Scientific American* column.[33] Since then, cryptography has become a widely used tool in communications, computer networks, and computer security generally.

Some modern cryptographic techniques can only keep their keys secret if certain mathematical problems are intractable, such as the integer factorization or the discrete logarithm problems, so there are deep connections with abstract mathematics. There are very few cryptosystems that are proven to be unconditionally secure. The one-time pad is one, and was proven to be so by Claude Shannon. There are a few important algorithms that have been proven secure under certain assumptions. For example, the infeasibility of factoring extremely large integers is the basis for believing that RSA is secure, and some other systems, but even so, proof of unbreakability is unavailable since the underlying mathematical problem remains open. In practice, these are widely used, and are believed unbreakable in practice by most competent observers. There are systems similar to RSA, such as one by Michael O. Rabin that are provably secure provided factoring $n = pq$ is impossible; it is quite unusable in practice. The discrete logarithm problem is the basis for believing some other cryptosystems are secure, and again, there are related, less practical systems that are provably secure relative to the solvability or insolvability discrete log problem.[34]

As well as being aware of cryptographic history, cryptographic algorithm and system designers must also sensibly consider probable future developments while working on their designs. For instance, continuous improvements in computer processing power have increased the scope of brute-force attacks, so when specifying key lengths, the required key lengths are similarly advancing.[35] The potential impact of quantum computing are already being considered by some cryptographic system designers developing post-quantum cryptography. The announced imminence of small implementations of these machines may be making the need for preemptive caution rather more than merely speculative.[6]

Modern cryptography

Prior to the early 20th century, cryptography was mainly concerned with linguistic and lexicographic patterns. Since then cryptography has broadened in scope, and now makes extensive use of mathematical subdisciplines, including information theory, computational complexity, statistics, combinatorics, abstract algebra, number theory, and finite mathematics.[36] Cryptography is also a branch of engineering, but an unusual one since it deals with active,

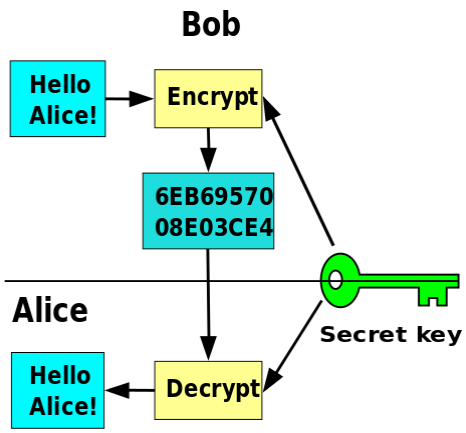
intelligent, and malevolent opposition; other kinds of engineering (e.g., civil or chemical engineering) need deal only with neutral natural forces. There is also active research examining the relationship between cryptographic problems and quantum physics.

Just as the development of digital computers and electronics helped in cryptanalysis, it made possible much more complex ciphers. Furthermore, computers allowed for the encryption of any kind of data representable in any binary format, unlike classical ciphers which only encrypted written language texts; this was new and significant. Computer use has thus supplanted linguistic cryptography, both for cipher design and cryptanalysis. Many computer ciphers can be characterized by their operation on binary bit sequences (sometimes in groups or blocks), unlike classical and mechanical schemes, which generally manipulate traditional characters (i.e., letters and digits) directly. However, computers have also assisted cryptanalysis, which has compensated to some extent for increased cipher complexity. Nonetheless, good modern ciphers have stayed ahead of cryptanalysis; it is typically the case that use of a quality cipher is very efficient (i.e., fast and requiring few resources, such as memory or CPU capability), while breaking it requires an effort many orders of magnitude larger, and vastly larger than that required for any classical cipher, making cryptanalysis so inefficient and impractical as to be effectively impossible.

Modern cryptography

Symmetric-key cryptography

Symmetric-key cryptography refers to encryption methods in which both the sender and receiver share the same key (or, less commonly, in which their keys are different, but related in an easily computable way). This was the only kind of encryption publicly known until June 1976.[32]



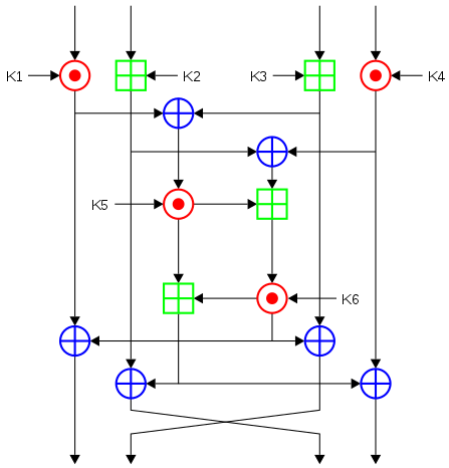
Symmetric-key cryptography, where a single key is used for encryption and decryption

Symmetric key ciphers are implemented as either block ciphers or stream ciphers. A block cipher enciphers input in blocks of plaintext as opposed to individual characters, the input form used by a stream cipher.

The Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) are block cipher designs that have been designated cryptography standards by the US government (though DES's designation was finally withdrawn after the AES was adopted).[37] Despite its deprecation as an official standard, DES (especially its still-approved and much more secure triple-DES variant) remains quite popular; it is used across a wide range of applications, from ATM encryption[38] to e-mail privacy[39] and secure remote access.[40] Many other block ciphers have been designed and released, with considerable variation in quality. Many, even some designed by capable practitioners, have been thoroughly broken, such as FEAL.[6][41]

Stream ciphers, in contrast to the 'block' type, create an arbitrarily long stream of key material, which is combined with the plaintext bit-by-bit or character-by-character, somewhat like the one-time pad. In a stream cipher, the output stream is created based on a hidden internal state that changes as the cipher operates. That internal state is initially set up using the secret key material. RC4 is a widely used stream cipher.[6] Block ciphers can be used as stream ciphers by generating blocks of a keystream (in place of a Pseudorandom number generator) and applying an XOR operation to each bit of the plaintext with each bit of the keystream.[42]

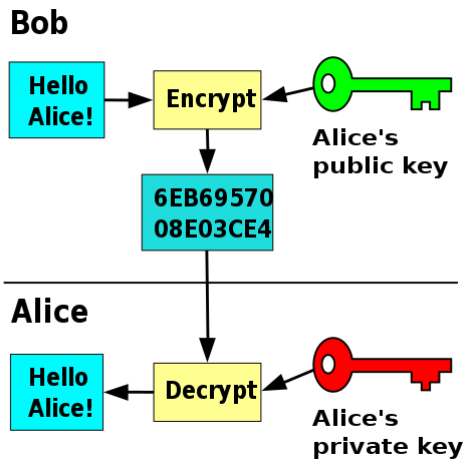
Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key can be used to authenticate the hash value upon receipt;[6][43] this additional complication blocks an attack scheme against bare digest algorithms, and so has been thought worth the effort. Cryptographic hash functions are a third type of cryptographic algorithm. They take a message of any length as input, and output a short, fixed-length hash, which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function that is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice. The US National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but is vulnerable to clashes as of 2011; and the US standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit."[44] Thus, a hash function design competition was meant to select a new U.S. national standard, to be called SHA-3, by 2012. The competition ended on October 2, 2012, when the NIST announced that Keccak would be the new SHA-3 hash algorithm.[45] Unlike block and stream ciphers that are invertible, cryptographic hash functions produce a hashed output that cannot be used to retrieve the original input data. Cryptographic hash functions are used to verify the authenticity of data retrieved from an untrusted source or to add a layer of security.



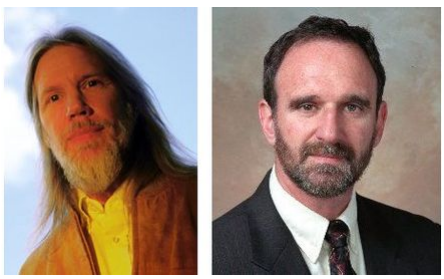
One round (out of 8.5) of the IDEA cipher, used in most versions of PGP and OpenPGP compatible software for time-efficient encryption of messages

Public-key cryptography

Symmetric-key cryptosystems use the same key for encryption and decryption of a message, although a message or group of messages can have a different key than others. A significant disadvantage of symmetric ciphers is the key management necessary to use them securely. Each distinct pair of communicating parties must, ideally, share a different key, and perhaps for each ciphertext exchanged as well. The number of keys required increases as the square of the number of network members, which very quickly requires complex key management schemes to keep them all consistent and secret.



Public-key cryptography, where different keys are used for encryption and decryption.


Whitfield Diffie and Martin Hellman, authors of the first published paper on public-key cryptography.

In a groundbreaking 1976 paper, Whitfield Diffie and Martin Hellman proposed the notion of *public-key* (also, more generally, called *asymmetric key*) cryptography in which two different but mathematically related keys are used—a *public* key and a *private* key.[46] A public key system is so constructed that calculation of one key (the 'private key') is computationally infeasible from the other (the 'public key'), even though they are necessarily related. Instead, both keys are generated

secretly, as an interrelated pair.[47] The historian David Kahn described public-key cryptography as "the most revolutionary new concept in the field since polyalphabetic substitution emerged in the Renaissance".[48]

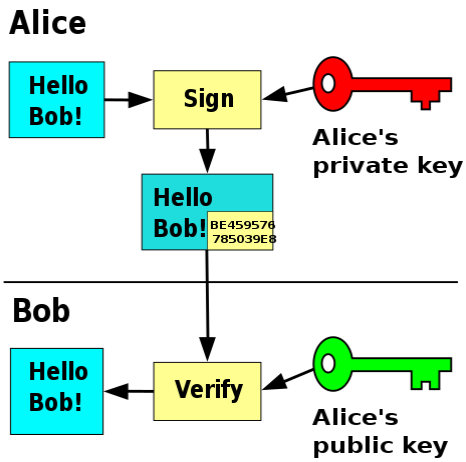
In public-key cryptosystems, the public key may be freely distributed, while its paired private key must remain secret. In a public-key encryption system, the *public key* is used for encryption, while the *private* or *secret key* is used for decryption. While Diffie and Hellman could not find such a system, they showed that public-key cryptography was indeed possible by presenting the Diffie–Hellman key exchange protocol, a solution that is now widely used in secure communications to allow two parties to secretly agree on a shared encryption key.[32] The X.509 standard defines the most commonly used format for public key certificates.[49]

Diffie and Hellman's publication sparked widespread academic efforts in finding a practical public-key encryption system. This race was finally won in 1978 by Ronald Rivest, Adi Shamir, and Len Adleman, whose solution has since become known as the RSA algorithm.[50]

The Diffie–Hellman and RSA algorithms, in addition to being the first publicly known examples of high-quality public-key algorithms, have been among the most widely used. Other asymmetric-key algorithms include the Cramer–Shoup cryptosystem, ElGamal encryption, and various elliptic curve techniques.

A document published in 1997 by the Government Communications Headquarters (GCHQ), a British intelligence organization, revealed that cryptographers at GCHQ had anticipated several academic developments.[51] Reportedly, around 1970, James H. Ellis had conceived the principles of asymmetric key cryptography. In 1973, Clifford Cocks invented a solution that was very similar in design rationale to RSA.[51][52] In 1974, Malcolm J. Williamson is claimed to have developed the Diffie–Hellman key exchange.[53]

Public-key cryptography is also used for implementing digital signature schemes. A digital signature is reminiscent of an ordinary signature; they both have the characteristic of being easy for a user to produce, but difficult for anyone else to forge. Digital signatures can also be permanently tied to the content of the message being signed; they cannot then be 'moved' from one document to another, for any attempt will be detectable. In digital signature schemes, there are two algorithms: one for *signing*, in which a secret key is used to process the message (or a hash of the message, or both), and one for *verification*, in which the matching public key is used with the message to check the validity of the signature. RSA and DSA are two of the most popular digital signature schemes. Digital signatures are central to the operation of public key infrastructures and many network security schemes (e.g., SSL/TLS, many VPNs, etc.).[41]



In this example the message is only signed and not encrypted. 1) Alice signs a message with her private key. 2) Bob can verify that Alice sent the message and that the message has not been modified.

Public-key algorithms are most often based on the computational complexity of "hard" problems, often from number theory. For example, the hardness of RSA is related to the integer factorization problem, while Diffie–Hellman and DSA are related to the discrete logarithm problem. The security of elliptic curve cryptography is based on number theoretic problems involving elliptic curves. Because of the difficulty of the underlying problems, most public-key algorithms involve operations such as modular multiplication and exponentiation, which are much more computationally expensive than the techniques used in most block ciphers, especially with typical key sizes. As a result, public-key cryptosystems are commonly hybrid

cryptosystems, in which a fast high-quality symmetric-key encryption algorithm is used for the message itself, while the relevant symmetric key is sent with the message, but encrypted using a public-key algorithm. Similarly, hybrid signature schemes are often used, in which a cryptographic hash function is computed, and only the resulting hash is digitally signed.[6]

Cryptographic Hash Functions

Cryptographic Hash Functions are cryptographic algorithms that are ways to generate and utilize specific keys to encrypt data for either symmetric or asymmetric encryption, and such functions may be viewed as keys themselves. They take a message of any length as input, and output a short, fixed-length hash, which can be used in (for example) a digital signature. For good hash functions, an attacker cannot find two messages that produce the same hash. MD4 is a long-used hash function that is now broken; MD5, a strengthened variant of MD4, is also widely used but broken in practice. The US National Security Agency developed the Secure Hash Algorithm series of MD5-like hash functions: SHA-0 was a flawed algorithm that the agency withdrew; SHA-1 is widely deployed and more secure than MD5, but cryptanalysts have identified attacks against it; the SHA-2 family improves on SHA-1, but is vulnerable to clashes as of 2011; and the US standards authority thought it "prudent" from a security perspective to develop a new standard to "significantly improve the robustness of NIST's overall hash algorithm toolkit."[44] Thus, a hash function design competition was meant to select a new U.S. national standard, to be called SHA-3, by 2012. The competition ended on October 2, 2012, when the NIST announced that Keccak would be the new SHA-3 hash algorithm.[45] Unlike block and stream ciphers that are invertible, cryptographic hash functions produce a hashed output that cannot be used to retrieve the original input data. Cryptographic hash functions are used to verify the authenticity of data retrieved from an untrusted source or to add a layer of security.

Cryptanalysis

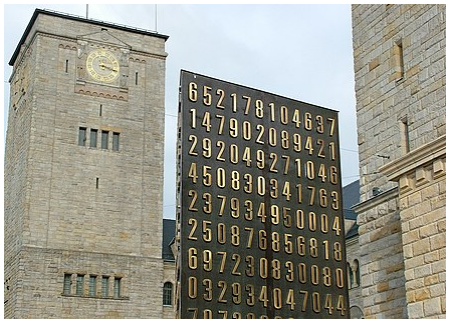
The goal of cryptanalysis is to find some weakness or insecurity in a cryptographic scheme, thus permitting its subversion or evasion.

It is a common misconception that every encryption method can be broken. In connection with his WWII work at Bell Labs, Claude Shannon proved that the one-time pad cipher is unbreakable, provided the key material is truly random, never reused, kept secret from all possible attackers, and of equal or greater length than the message.[54] Most ciphers, apart from the one-time pad, can be broken with enough computational effort by brute force attack, but the amount of effort needed may be exponentially dependent on the key size, as compared to the effort needed to make use of the cipher. In such cases, effective security could be achieved if it is proven that the effort required (i.e., "work factor", in Shannon's terms) is beyond the ability of any adversary. This means it must be shown that no efficient method (as opposed to the time-consuming brute force method) can be found to break the cipher. Since no such proof has been found to date, the one-time-pad remains the only theoretically unbreakable cipher. Although well-implemented one-time-pad encryption cannot be broken, traffic analysis is still possible.

There are a wide variety of cryptanalytic attacks, and they can be classified in any of several ways. A common distinction turns on what Eve (an attacker) knows and what capabilities are available. In a ciphertext-only attack, Eve has access only to the ciphertext (good modern cryptosystems are usually effectively immune to ciphertext-only attacks). In a known-plaintext attack, Eve has access to a ciphertext and its corresponding plaintext (or to many such pairs). In a chosen-plaintext attack, Eve may choose a plaintext and learn its corresponding ciphertext (perhaps many times); an example is gardening, used by the British during WWII. In a chosen-ciphertext attack, Eve may be able to *choose* ciphertexts and learn their corresponding plaintexts.[6] Finally in a man-in-the-middle attack Eve gets in between Alice (the sender) and Bob (the recipient), accesses and modifies

Variants of the Enigma machine, used by Germany's military and civil authorities from the late 1920s through World War II, implemented a complex electro-mechanical polyalphabetic cipher. Breaking and reading of the Enigma cipher at Poland's Cipher Bureau, for 7 years before the war, and subsequent decryption at Bletchley Park, was important to Allied victory.[14]

the traffic and then forwards it to the recipient.[55] Also important, often overwhelmingly so, are mistakes (generally in the design or use of one of the protocols involved).



Poznań monument (*center*) to Polish cryptanalysts whose breaking of Germany's Enigma machine ciphers, beginning in 1932, altered the course of World War II

Cryptanalysis of symmetric-key ciphers typically involves looking for attacks against the block ciphers or stream ciphers that are more efficient than any attack that could be against a perfect cipher. For example, a simple brute force attack against DES requires one known plaintext and 2^{55} decryptions, trying approximately half of the possible keys, to reach a point at which chances are better than even that the key sought will have been found. But this may not be enough assurance; a linear cryptanalysis attack against DES requires 2^{43} known plaintexts (with their corresponding ciphertexts) and approximately 2^{43} DES operations.[56] This is a considerable improvement over brute force attacks.

Public-key algorithms are based on the computational difficulty of various problems. The most famous of these are the difficulty of integer factorization of semiprimes and the difficulty of calculating discrete logarithms, both of which are not yet proven to be solvable in polynomial time (**P**) using only a classical Turing-complete computer. Much public-key cryptanalysis concerns designing algorithms in **P** that can solve these problems, or using other technologies, such as quantum computers. For instance, the best-known algorithms for solving the elliptic curve-based version of discrete logarithm are much more time-consuming than the best-known algorithms for factoring, at least for problems of more or less equivalent size. Thus, to achieve an equivalent strength of encryption, techniques that depend upon the difficulty of factoring large composite numbers, such as the RSA cryptosystem, require larger keys than elliptic curve techniques. For this reason, public-key cryptosystems based on elliptic curves have become popular since their invention in the mid-1990s.

While pure cryptanalysis uses weaknesses in the algorithms themselves, other attacks on cryptosystems are based on actual use of the algorithms in real devices, and are called *side-channel attacks*. If a cryptanalyst has access to, for example, the amount of time the device took to encrypt a number of plaintexts or report an error in a password or PIN character, he may be able to use a timing attack to break a cipher that is otherwise resistant to analysis. An attacker might also study the pattern and length of messages to derive valuable information; this is known as traffic analysis[57] and can be quite useful to an alert adversary. Poor administration of a cryptosystem, such as permitting too short keys, will make any system vulnerable, regardless of other virtues. Social engineering and other attacks against humans (e.g., bribery, extortion, blackmail, espionage, torture, ...) are usually employed due to being more cost-effective and feasible to perform in a reasonable amount of time compared to pure cryptanalysis by a high margin.

Cryptographic primitives

Much of the theoretical work in cryptography concerns cryptographic *primitives*—algorithms with basic cryptographic properties—and their relationship to other cryptographic problems. More complicated cryptographic tools are then built from these basic primitives. These primitives provide fundamental properties, which are used to develop more complex tools called *cryptosystems* or *cryptographic protocols*, which guarantee one or more high-level security properties. Note, however, that the distinction between cryptographic *primitives* and cryptosystems, is quite arbitrary; for example, the RSA algorithm is sometimes considered a cryptosystem, and sometimes a primitive. Typical examples of cryptographic primitives include pseudorandom functions, one-way functions, etc.

Cryptosystems

One or more cryptographic primitives are often used to develop a more complex algorithm, called a cryptographic system, or *cryptosystem*. Cryptosystems (e.g., El-Gamal encryption) are designed to provide particular functionality (e.g., public key encryption) while guaranteeing certain security properties (e.g., chosen-plaintext attack (CPA) security in the random oracle model). Cryptosystems use the properties of the underlying cryptographic primitives to support the system's security properties. As the distinction between primitives and cryptosystems is somewhat arbitrary, a sophisticated cryptosystem can be derived from a combination of several more primitive cryptosystems. In many cases, the cryptosystem's structure involves back and forth communication among two or more parties in space (e.g., between the sender of a secure message and its receiver) or across time (e.g., cryptographically protected backup data). Such cryptosystems are sometimes called *cryptographic protocols*.

Some widely known cryptosystems include RSA, Schnorr signature, ElGamal encryption, and Pretty Good Privacy (PGP). More complex cryptosystems include electronic cash[58] systems, signcryption systems, etc. Some more 'theoretical' cryptosystems include interactive proof systems,[59] (like zero-knowledge proofs),[60] systems for secret sharing,[61][62] etc.

Lightweight cryptography

Lightweight cryptography (LWC) concerns cryptographic algorithms developed for a strictly constrained environment. The growth of Internet of Things (IoT) has spiked research into the development of lightweight algorithms that are better suited for the environment. An IoT environment requires strict constraints on power consumption, processing power, and security.[63] Algorithms such as PRESENT, AES, and SPECK are examples of the many LWC algorithms that have been developed to achieve the standard set by the National Institute of Standards and Technology.[64]

Applications

General

Cryptography is widely used on the internet to help protect user-data and prevent eavesdropping. To ensure secrecy during transmission, many systems use private key cryptography to protect transmitted information. With public-key systems, one can maintain secrecy without a master key or a large number of keys.[65] But, some algorithms like Bitlocker and Veracrypt are generally not private-public key cryptography. Such as Veracrypt, it uses a password hash to generate the single private key. However, it can be configured to run in public-private key systems. The C++ opensource encryption library OpenSSL provides free and opensource encryption software and tools. The most commonly used encryption cipher suit is AES,[66] as it has hardware acceleration

for all x86 based processors that has AES-NI. A close contender is ChaCha20-Poly1305, which is a stream cipher, however it is commonly used for mobile devices as they are ARM based which does not feature AES-NI instruction set extension.

Cybersecurity

Cryptography can be used to secure communications by encrypting them. Websites use encryption via HTTPS.[67] "End-to-end" encryption, where only sender and receiver can read messages, is implemented for email in Pretty Good Privacy and for secure messaging in general in WhatsApp, Signal and Telegram.[67]

Operating systems use encryption to keep passwords secret, conceal parts of the system, and ensure that software updates are truly from the system maker.[67] Instead of storing plaintext passwords, computer systems store hashes thereof; then, when a user logs in, the system passes the given password through a cryptographic hash function and compares it to the hashed value on file. In this manner, neither the system nor an attacker has at any point access to the password in plaintext.[67]

Encryption is sometimes used to encrypt one's entire drive. For example, University College London has implemented BitLocker (a program by Microsoft) to render drive data opaque without users logging in.[67]

Cryptocurrencies and cryptoeconomics

Cryptographic techniques enable cryptocurrency technologies, such as distributed ledger technologies (e.g., blockchains), which finance cryptoeconomics applications such as decentralized finance (DeFi).[68][69] Key cryptographic techniques that enable cryptocurrencies and cryptoeconomics include, but are not limited to: cryptographic keys, cryptographic hash functions, asymmetric (public key) encryption, Multi-Factor Authentication (MFA), End-to-End Encryption (E2EE), and Zero Knowledge Proofs (ZKP).[69]

Legal issues

Prohibitions

Cryptography has long been of interest to intelligence gathering and law enforcement agencies.[10] Secret communications may be criminal or even treasonous. Because of its facilitation of privacy, and the diminution of privacy attendant on its prohibition, cryptography is also of considerable interest to civil rights supporters. Accordingly, there has been a history of controversial legal issues surrounding cryptography, especially since the advent of inexpensive computers has made widespread access to high-quality cryptography possible.

In some countries, even the domestic use of cryptography is, or has been, restricted. Until 1999, France significantly restricted the use of cryptography domestically, though it has since relaxed many of these rules. In China and Iran, a license is still required to use cryptography.[8] Many countries have tight restrictions on the use of cryptography. Among the more restrictive are laws in Belarus, Kazakhstan, Mongolia, Pakistan, Singapore, Tunisia, and Vietnam.[70]

In the United States, cryptography is legal for domestic use, but there has been much conflict over legal issues related to cryptography.[10] One particularly important issue has been the export of cryptography and cryptographic software and hardware. Probably because of the importance of

cryptanalysis in World War II and an expectation that cryptography would continue to be important for national security, many Western governments have, at some point, strictly regulated export of cryptography. After World War II, it was illegal in the US to sell or distribute encryption technology overseas; in fact, encryption was designated as auxiliary military equipment and put on the United States Munitions List.[71] Until the development of the personal computer, asymmetric key algorithms (i.e., public key techniques), and the Internet, this was not especially problematic. However, as the Internet grew and computers became more widely available, high-quality encryption techniques became well known around the globe.

Export controls

In the 1990s, there were several challenges to US export regulation of cryptography. After the source code for Philip Zimmermann's Pretty Good Privacy (PGP) encryption program found its way onto the Internet in June 1991, a complaint by RSA Security (then called RSA Data Security, Inc.) resulted in a lengthy criminal investigation of Zimmermann by the US Customs Service and the FBI, though no charges were ever filed.[72][73] Daniel J. Bernstein, then a graduate student at UC Berkeley, brought a lawsuit against the US government challenging some aspects of the restrictions based on free speech grounds. The 1995 case *Bernstein v. United States* ultimately resulted in a 1999 decision that printed source code for cryptographic algorithms and systems was protected as free speech by the United States Constitution.[74]

In 1996, thirty-nine countries signed the Wassenaar Arrangement, an arms control treaty that deals with the export of arms and "dual-use" technologies such as cryptography. The treaty stipulated that the use of cryptography with short key-lengths (56-bit for symmetric encryption, 512-bit for RSA) would no longer be export-controlled.[75] Cryptography exports from the US became less strictly regulated as a consequence of a major relaxation in 2000;[76] there are no longer very many restrictions on key sizes in US-exported mass-market software. Since this relaxation in US export restrictions, and because most personal computers connected to the Internet include US-sourced web browsers such as Firefox or Internet Explorer, almost every Internet user worldwide has potential access to quality cryptography via their browsers (e.g., via Transport Layer Security). The Mozilla Thunderbird and Microsoft Outlook E-mail client programs similarly can transmit and receive emails via TLS, and can send and receive email encrypted with S/MIME. Many Internet users don't realize that their basic application software contains such extensive cryptosystems. These browsers and email programs are so ubiquitous that even governments whose intent is to regulate civilian use of cryptography generally don't find it practical to do much to control distribution or use of cryptography of this quality, so even when such laws are in force, actual enforcement is often effectively impossible.

NSA involvement


NSA headquarters in Fort Meade, Maryland

Another contentious issue connected to cryptography in the United States is the influence of the National Security Agency on cipher development and policy.[10] The NSA was involved with the design of DES during its development at IBM and its consideration by the National Bureau of Standards as a possible Federal Standard for cryptography.[77] DES was designed to be resistant to differential cryptanalysis,[78] a powerful and general cryptanalytic technique known to the NSA and IBM, that became publicly known only when it was rediscovered in the late 1980s.[79] According to Steven Levy, IBM discovered differential cryptanalysis,[73] but kept the technique secret at the NSA's request. The technique became

publicly known only when Biham and Shamir re-discovered and announced it some years later. The entire affair illustrates the difficulty of determining what resources and knowledge an attacker might actually have.

Another instance of the NSA's involvement was the 1993 Clipper chip affair, an encryption microchip intended to be part of the Capstone cryptography-control initiative. Clipper was widely criticized by cryptographers for two reasons. The cipher algorithm (called Skipjack) was then classified (declassified in 1998, long after the Clipper initiative lapsed). The classified cipher caused concerns that the NSA had deliberately made the cipher weak in order to assist its intelligence efforts. The whole initiative was also criticized based on its violation of Kerckhoffs's Principle, as the scheme included a special escrow key held by the government for use by law enforcement (i.e. wiretapping).[73]

Digital rights management

Cryptography is central to digital rights management (DRM), a group of techniques for technologically controlling use of copyrighted material, being widely implemented and deployed at the behest of some copyright holders. In 1998, U.S. President Bill Clinton signed the Digital Millennium Copyright Act (DMCA), which criminalized all production, dissemination, and use of certain cryptanalytic techniques and technology (now known or later discovered); specifically, those that could be used to circumvent DRM technological schemes.[80] This had a noticeable impact on the cryptography research community since an argument can be made that any cryptanalytic research violated the DMCA. Similar statutes have since been enacted in several countries and regions, including the implementation in the EU Copyright Directive. Similar restrictions are called for by treaties signed by World Intellectual Property Organization member-states.

The United States Department of Justice and FBI have not enforced the DMCA as rigorously as had been feared by some, but the law, nonetheless, remains a controversial one. Niels Ferguson, a well-respected cryptography researcher, has publicly stated that he will not release some of his research into an Intel security design for fear of prosecution under the DMCA.[81] Cryptologist Bruce Schneier has argued that the DMCA encourages vendor lock-in, while inhibiting actual measures toward cyber-security.[82] Both Alan Cox (longtime Linux kernel developer) and Edward Felten (and some of his students at Princeton) have encountered problems related to the Act. Dmitry Sklyarov was arrested during a visit to the US from Russia, and jailed for five months pending trial for alleged violations of the DMCA arising from work he had done in Russia, where the work was legal. In 2007, the cryptographic keys responsible for Blu-ray and HD DVD content scrambling were discovered and released onto the Internet. In both cases, the Motion Picture Association of America sent out numerous DMCA takedown notices, and there was a massive Internet backlash[11] triggered by the perceived impact of such notices on fair use and free speech.

Forced disclosure of encryption keys

In the United Kingdom, the Regulation of Investigatory Powers Act gives UK police the powers to force suspects to decrypt files or hand over passwords that protect encryption keys. Failure to comply is an offense in its own right, punishable on conviction by a two-year jail sentence or up to five years in cases involving national security.[9] Successful prosecutions have occurred under the Act; the first, in 2009,[83] resulted in a term of 13 months' imprisonment.[84] Similar forced disclosure laws in Australia, Finland, France, and India compel individual suspects under investigation to hand over encryption keys or passwords during a criminal investigation.

In the United States, the federal criminal case of *United States v. Fricosu* addressed whether a search warrant can compel a person to reveal an encryption passphrase or password.[85] The Electronic Frontier Foundation (EFF) argued that this is a violation of the protection from self-incrimination given by the Fifth Amendment.[86] In 2012, the court ruled that under the All Writs Act, the defendant was required to produce an unencrypted hard drive for the court.[87]

In many jurisdictions, the legal status of forced disclosure remains unclear.

The 2016 FBI–Apple encryption dispute concerns the ability of courts in the United States to compel manufacturers' assistance in unlocking cell phones whose contents are cryptographically protected.

As a potential counter-measure to forced disclosure some cryptographic software supports plausible deniability, where the encrypted data is indistinguishable from unused random data (for example such as that of a drive which has been securely wiped).

See also

- Collision attack
- Comparison of cryptography libraries
- Crypto Wars – Attempts to limit access to strong cryptography
- Encyclopedia of Cryptography and Security – Book by Technische Universiteit Eindhoven
- Global surveillance – Mass surveillance across national borders
- Indistinguishability obfuscation – Type of cryptographic software obfuscation
- Information theory – Scientific study of digital information
- Outline of cryptography – Overview of and topical guide to cryptography
  - List of cryptographers
  - List of important publications in cryptography
  - List of multiple discoveries
  - List of unsolved problems in computer science
- Secure cryptoprocessor
- Strong cryptography – Term applied to cryptographic systems that are highly resistant to cryptanalysis
- Syllabical and Steganographical Table – Eighteenth-century work believed to be the first cryptography chart – first cryptography chart
- World Wide Web Consortium's Web Cryptography API – World Wide Web Consortium cryptography standard

References

1. Liddell, Henry George; Scott, Robert; Jones, Henry Stuart; McKenzie, Roderick (1984). *A Greek-English Lexicon*. Oxford University Press.
2. Rivest, Ronald L. (1990). "Cryptography". In J. Van Leeuwen (ed.). *Handbook of Theoretical Computer Science*. Vol. 1. Elsevier.
3. Bellare, Mihir; Rogaway, Phillip (21 September 2005). "Introduction". *Introduction to Modern Cryptography*. p. 10.
4. Sadkhan, Sattar B. (December 2013). "Key note lecture multidisciplinary in cryptology and information security" (https://ieeexplore.ieee.org/abstract/document/6998773). *2013 International Conference on Electrical Communication, Computer, Power, and Control Engineering (ICECCPCE)*: 1–2. doi:10.1109/ICECCPCE.2013.6998773 (https://doi.org/10.1109%2FICECCPCE.2013.6998773).

5. "Crypto FAQ: What is cryptography & how does it work?" (https://cryptographyworks.com/faq/what-is-cryptography.html). *Cryptography Works™*. Retrieved 27 August 2022.
6. Menezes, A.J.; van Oorschot, P.C.; Vanstone, S.A. (1997). *Handbook of Applied Cryptography* (https://archive.org/details/handbookofapplie0000mene). ISBN 978-0-8493-8523-0.
7. Biggs, Norman (2008). *Codes: An introduction to Information Communication and Cryptography* (https://archive.org/details/codesintroductio00bigg_911). Springer. p. 171 (https://archive.org/details/codesintroductio00bigg_911/page/n176).
8. "Overview per country" (http://www.cryptolaw.org/cls2.htm). *Crypto Law Survey*. February 2013. Retrieved 26 March 2015.
9. "UK Data Encryption Disclosure Law Takes Effect" (https://web.archive.org/web/20120120135135/http://www.pcworld.com/article/137881/uk_data_encryption_disclosure_law_takes_effect.html). *PC World*. 1 October 2007. Archived from the original (http://www.pcworld.com/article/137881/uk_data_encryption_disclosure_law_takes_effect.html) on 20 January 2012. Retrieved 26 March 2015.
10. Ranger, Steve (24 March 2015). "The undercover war on your internet secrets: How online surveillance cracked our trust in the web" (https://web.archive.org/web/20160612190952/http://www.techrepublic.com/article/the-undercover-war-on-your-internet-secrets-how-online-surveillance-cracked-our-trust-in-the-web/). TechRepublic. Archived from the original (https://www.techrepublic.com/article/the-undercover-war-on-your-internet-secrets-how-online-surveillance-cracked-our-trust-in-the-web/) on 12 June 2016. Retrieved 12 June 2016.
11. Doctorow, Cory (2 May 2007). "Digg users revolt over AACS key" (https://boingboing.net/2007/05/02/digg-users-revolt-ov.html). *Boing Boing*. Retrieved 26 March 2015.
12. Whalen, Terence (1994). "The Code for Gold: Edgar Allan Poe and Cryptography". *Representations*. University of California Press. **46** (46): 35–57. doi:10.2307/2928778 (https://doi.org/10.2307%2F2928778). JSTOR 2928778 (https://www.jstor.org/stable/2928778).
13. Rosenheim 1997, p. 20
14. Kahn, David (1967). *The Codebreakers*. ISBN 978-0-684-83130-5.
15. "An Introduction to Modern Cryptosystems" (https://www.giac.org/paper/gsec/2604/introduction-modern-cryptosystems/104482).
16. Sharbaf, M.S. (1 November 2011). "Quantum cryptography: An emerging technology in network security". *2011 IEEE International Conference on Technologies for Homeland Security (HST)*: 13–19. doi:10.1109/THS.2011.6107841 (https://doi.org/10.1109%2FTHS.2011.6107841). ISBN 978-1-4577-1376-7. S2CID 17915038 (https://api.semanticscholar.org/CorpusID:17915038).
17. "cryptology | Britannica" (https://www.britannica.com/topic/cryptology). *www.britannica.com*. Retrieved 22 June 2022.
18. Oded Goldreich, *Foundations of Cryptography, Volume 1: Basic Tools*, Cambridge University Press, 2001, ISBN 0-521-79172-3
19. "Cryptology (definition)" (http://www.merriam-webster.com/dictionary/cryptology). *Merriam-Webster's Collegiate Dictionary* (11th ed.). Merriam-Webster. Retrieved 26 March 2015.
20. Shirey, Rob (May 2000). "Internet Security Glossary" (https://tools.ietf.org/html/rfc2828). *Internet Engineering Task Force*. RFC 2828 (https://tools.ietf.org/html/rfc2828). Retrieved 26 March 2015.
21. Saltzman, Benjamin A. (1 October 2018). "Vt hkskdkxt: Early Medieval Cryptography, Textual Errors, and Scribal Agency" (https://www.journals.uchicago.edu/doi/10.1086/698861). *Speculum*. **93** (4): 975–1009. doi:10.1086/698861 (https://doi.org/10.1086%2F698861). ISSN 0038-7134 (https://www.worldcat.org/issn/0038-7134). S2CID 165362817 (https://api.semanticscholar.org/CorpusID:165362817).
22. lAshchenko, V.V. (2002). *Cryptography: an introduction* (https://books.google.com/books?id=cH-NGrpcIMcC&pg=PA6). AMS Bookstore. p. 6. ISBN 978-0-8218-2986-8.
23. electricpulp.com. "CODES – Encyclopaedia Iranica" (http://www.iranicaonline.org/articles/codes-romuz-sg). *www.iranicaonline.org*.

24. Kahn, David (1996). *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet* (https://books.google.com/books?id=3S8rhOEmDIIC&q=david+kahn+the+codebreakers). Simon and Schuster. ISBN 9781439103555.
25. Broemeling, Lyle D. (1 November 2011). "An Account of Early Statistical Inference in Arab Cryptology". *The American Statistician*. **65** (4): 255–257. doi:10.1198/tas.2011.10191 (https://doi.org/10.1198%2Ftas.2011.10191). S2CID 123537702 (https://api.semanticscholar.org/CorpusID:123537702).
26. Singh, Simon (2000). *The Code Book*. New York: Anchor Books. pp. 14–20. ISBN 978-0-385-49532-5.
27. Al-Kadi, Ibrahim A. (April 1992). "The origins of cryptology: The Arab contributions". *Cryptologia*. **16** (2): 97–126. doi:10.1080/0161-119291866801 (https://doi.org/10.1080%2F0161-119291866801).
28. Schrödel, Tobias (October 2008). "Breaking Short Vigenère Ciphers". *Cryptologia*. **32** (4): 334–337. doi:10.1080/01611190802336097 (https://doi.org/10.1080%2F01611190802336097). S2CID 21812933 (https://api.semanticscholar.org/CorpusID:21812933).
29. Hakim, Joy (1995). *A History of US: War, Peace and all that Jazz*. New York: Oxford University Press. ISBN 978-0-19-509514-2.
30. Gannon, James (2001). *Stealing Secrets, Telling Lies: How Spies and Codebreakers Helped Shape the Twentieth Century* (https://archive.org/details/stealingsecretst00gann). Washington, D.C.: Brassey's. ISBN 978-1-57488-367-1.
31. "The Legacy of DES - Schneier on Security" (https://www.schneier.com/blog/archives/2004/10/the_legacy_of_d.html). *www.schneier.com*. Retrieved 26 January 2022.
32. Diffie, Whitfield; Hellman, Martin (November 1976). "New Directions in Cryptography" (http://www-ee.stanford.edu/~hellman/publications/24.pdf) (PDF). *IEEE Transactions on Information Theory*. IT-22 (6): 644–654. CiteSeerX 10.1.1.37.9720 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.9720). doi:10.1109/tit.1976.1055638 (https://doi.org/10.1109%2Ftit.1976.1055638).
33. Singh, Simon (1999). *The Code Book: The Science of Secrecy From Ancient Egypt To Quantum Cryptography* (https://archive.org/details/codebook00simo/page/278) (First Anchor Books ed.). New York: Anchor Books. pp. 278 (https://archive.org/details/codebook00simo/page/278). ISBN 978-0-385-49532-5.
34. *Cryptography: Theory and Practice*, Third Edition (Discrete Mathematics and Its Applications), 2005, by Douglas R. Stinson, Chapman and Hall/CRC
35. Blaze, Matt; Diffie, Whitefield; Rivest, Ronald L.; Schneier, Bruce; Shimomura, Tsutomu; Thompson, Eric; Wiener, Michael (January 1996). "Minimal key lengths for symmetric ciphers to provide adequate commercial security" (http://www.fortify.net/related/cryptographers.html). Fortify. Retrieved 26 March 2015.
36. Diffie, W.; Hellman, M. (1 September 2006). "New directions in cryptography" (https://dl.acm.org/doi/10.1109/TIT.1976.1055638). *IEEE Transactions on Information Theory*. **22** (6): 644–654. doi:10.1109/TIT.1976.1055638 (https://doi.org/10.1109%2FTIT.1976.1055638).
37. "FIPS PUB 197: The official Advanced Encryption Standard" (https://web.archive.org/web/20150407153905/http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf) (PDF). *Computer Security Resource Center*. National Institute of Standards and Technology. Archived from the original (http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf) (PDF) on 7 April 2015. Retrieved 26 March 2015.
38. "NCUA letter to credit unions" (https://www.ncua.gov/Resources/Documents/LCU2004-09.pdf) (PDF). *National Credit Union Administration*. July 2004. Retrieved 26 March 2015.
39. Finney, Hal; Thayer, Rodney L.; Donnerhacke, Lutz; Callas, Jon (November 1998). "Open PGP Message Format" (https://tools.ietf.org/html/rfc2440). *Internet Engineering Task Force*. RFC 2440 (https://tools.ietf.org/html/rfc2440). Retrieved 26 March 2015.
40. Golen, Pawel (19 July 2002). "SSH" (http://www.windowsecurity.com/articles/SSH.html). *WindowSecurity*. Retrieved 26 March 2015.

41. Schneier, Bruce (1996). *Applied Cryptography* (https://archive.org/details/Applied_Cryptography_2nd_ed._B._Schneier) (2nd ed.). Wiley. ISBN 978-0-471-11709-4.
42. Paar, Christof (2009). *Understanding cryptography : a textbook for students and practitioners* (https://www.worldcat.org/oclc/567365751). Jan Pelzl. Berlin: Springer. p. 123. ISBN 978-3-642-04101-3. OCLC 567365751 (https://www.worldcat.org/oclc/567365751).
43. Bernstein, Daniel J.; Lange, Tanja (14 September 2017). "Post-quantum cryptography" (http://www.nature.com/articles/nature23461). *Nature*. **549** (7671): 188–194. doi:10.1038/nature23461 (https://doi.org/10.1038%2Fnature23461). ISSN 0028-0836 (https://www.worldcat.org/issn/0028-0836).
44. "Notices". *Federal Register*. **72** (212). 2 November 2007. "Archived copy" (https://web.archive.org/web/20080228075550/http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf) (PDF). Archived from the original on 28 February 2008. Retrieved 27 January 2009.
45. "NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition" (https://www.nist.gov/itl/csd/sha-100212.cfm). *Tech Beat*. National Institute of Standards and Technology. 2 October 2012. Retrieved 26 March 2015.
46. Diffie, Whitfield; Hellman, Martin (8 June 1976). "Multi-user cryptographic techniques". *AFIPS Proceedings*. **45**: 109–112. doi:10.1145/1499799.1499815 (https://doi.org/10.1145%2F1499799.1499815). S2CID 13210741 (https://api.semanticscholar.org/CorpusID:13210741).
47. Ralph Merkle was working on similar ideas at the time and encountered publication delays, and Hellman has suggested that the term used should be Diffie–Hellman–Merkle aysmmetric key cryptography.
48. Kahn, David (Fall 1979). "Cryptology Goes Public". *Foreign Affairs*. **58** (1): 141–159. doi:10.2307/20040343 (https://doi.org/10.2307%2F20040343). JSTOR 20040343 (https://www.jstor.org/stable/20040343).
49. "Using Client-Certificate based authentication with NGINX on Ubuntu - SSLTrust" (https://www.ssltrust.com.au/help/setup-guides/client-certificate-authentication). *SSLTrust*. Retrieved 13 June 2019.
50. Rivest, Ronald L.; Shamir, A.; Adleman, L. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". *Communications of the ACM*. **21** (2): 120–126. CiteSeerX 10.1.1.607.2677 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.607.2677). doi:10.1145/359340.359342 (https://doi.org/10.1145%2F359340.359342). S2CID 2873616 (https://api.semanticscholar.org/CorpusID:2873616).
"Archived copy" (https://web.archive.org/web/20011116122233/http://theory.lcs.mit.edu/~rivest/rsapaper.pdf) (PDF). Archived from the original (http://theory.lcs.mit.edu/~rivest/rsapaper.pdf) (PDF) on 16 November 2001. Retrieved 20 April 2006.
Previously released as an MIT "Technical Memo" in April 1977, and published in Martin Gardner's *Scientific American* Mathematical recreations column
51. Wayner, Peter (24 December 1997). "British Document Outlines Early Encryption Discovery" (https://www.nytimes.com/library/cyber/week/122497encrypt.html). *The New York Times*. Retrieved 26 March 2015.
52. Cocks, Clifford (20 November 1973). "A Note on 'Non-Secret Encryption'" (http://www.fi.muni.cz/usr/matyas/lecture/paper2.pdf) (PDF). *CESG Research Report*.
53. Singh, Simon (1999). *The Code Book* (https://archive.org/details/codebookevolutio00sing). Doubleday. pp. 279–292 (https://archive.org/details/codebookevolutio00sing/page/279). ISBN 9780385495318.
54. Shannon, Claude; Weaver, Warren (1963). *The Mathematical Theory of Communication*. University of Illinois Press. ISBN 978-0-252-72548-7.
55. "An Example of a Man-in-the-middle Attack Against Server Authenticated SSL-sessions" (http://www8.cs.umu.se/education/examina/Rapporter/MattiasEriksson.pdf) (PDF).

56. Junod, Pascal (2001). *On the Complexity of Matsui's Attack* (http://citeseer.ist.psu.edu/cache/papers/cs/22094/http:zSzzSzeprint.iacr.orgzSz2001zSz056.pdf/junod01complexity.pdf) (PDF). *Selected Areas in Cryptography*. Lecture Notes in Computer Science. Vol. 2259. pp. 199–211. doi:10.1007/3-540-45537-X_16 (https://doi.org/10.1007%2F3-540-45537-X_16). ISBN 978-3-540-43066-7.
57. Song, Dawn; Wagner, David A.; Tian, Xuqing (2001). "Timing Analysis of Keystrokes and Timing Attacks on SSH" (http://citeseer.ist.psu.edu/cache/papers/cs/22094/http:zSzzSzeprint.iacr.orgzSz2001zSz056.pdf/junod01complexity.pdf) (PDF). *Tenth USENIX Security Symposium*.
58. Brands, S. (1994). "Untraceable Off-line Cash in Wallet with Observers". *Untraceable Off-line Cash in Wallets with Observers* (https://web.archive.org/web/20110726214409/http://ftp.se.kde.org/pub/security/docs/ecash/crypto93.ps.gz). *Advances in Cryptology—Proceedings of CRYPTO*. Lecture Notes in Computer Science. Vol. 773. pp. 302–318. doi:10.1007/3-540-48329-2_26 (https://doi.org/10.1007%2F3-540-48329-2_26). ISBN 978-3-540-57766-9. Archived from the original (http://ftp.se.kde.org/pub/security/docs/ecash/crypto93.ps.gz) on 26 July 2011.
59. Babai, László (1985). "Trading group theory for randomness". *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85* (http://portal.acm.org/citation.cfm?id=22192). *Proceedings of the Seventeenth Annual Symposium on the Theory of Computing*. Stoc '85. pp. 421–429. CiteSeerX 10.1.1.130.3397 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.3397). doi:10.1145/22145.22192 (https://doi.org/10.1145%2F22145.22192). ISBN 978-0-89791-151-1. S2CID 17981195 (https://api.semanticscholar.org/CorpusID:17981195).
60. Goldwasser, S.; Micali, S.; Rackoff, C. (1989). "The Knowledge Complexity of Interactive Proof Systems". *SIAM Journal on Computing*. **18** (1): 186–208. CiteSeerX 10.1.1.397.4002 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.397.4002). doi:10.1137/0218012 (https://doi.org/10.1137%2F0218012).
61. Blakley, G. (June 1979). "Safeguarding cryptographic keys". *Proceedings of AFIPS 1979*. **48**: 313–317.
62. Shamir, A. (1979). "How to share a secret". *Communications of the ACM*. **22** (11): 612–613. doi:10.1145/359168.359176 (https://doi.org/10.1145%2F359168.359176). S2CID 16321225 (https://api.semanticscholar.org/CorpusID:16321225).
63. Gunathilake, Nilupulee A.; Al-Dubai, Ahmed; Buchana, William J. (2 November 2020). "Recent Advances and Trends in Lightweight Cryptography for IoT Security" (https://ieeexplore.ieee.org/document/9269083). *2020 16th International Conference on Network and Service Management (CNSM)*. Izmir, Turkey: IEEE: 1–5. doi:10.23919/CNSM50824.2020.9269083 (https://doi.org/10.23919%2FCNSM50824.2020.9269083). ISBN 978-3-903176-31-7. S2CID 227277538 (https://api.semanticscholar.org/CorpusID:227277538).
64. Thakor, Vishal A.; Razzaque, Mohammad Abdur; Khandaker, Muhammad R. A. (2021). "Lightweight Cryptography Algorithms for Resource-Constrained IoT Devices: A Review, Comparison and Research Opportunities" (https://doi.org/10.1109%2FACCESS.2021.3052867). *IEEE Access*. **9**: 28177–28193. doi:10.1109/ACCESS.2021.3052867 (https://doi.org/10.1109%2FACCESS.2021.3052867). ISSN 2169-3536 (https://www.worldcat.org/issn/2169-3536). S2CID 232042514 (https://api.semanticscholar.org/CorpusID:232042514).
65. Cohen, Fred (1995). "2.4 - Applications of Cryptography" (http://all.net/edu/curr/ip/Chap2-4.html). *all.net*. Archived (https://web.archive.org/web/19990824044441/http://all.net/edu/curr/ip/Chap2-4.html) from the original on 24 August 1999. Retrieved 21 December 2021.
66. "4 Common Encryption Methods to Shield Sensitive Data From Prying Eyes" (https://www.getapp.com/resources/common-encryption-methods/). *GetApp*. Retrieved 14 May 2022.

67. Chamberlain, Austin (12 March 2017). "Applications of Cryptography | UCL Risky Business" (https://blogs.ucl.ac.uk/infosec/2017/03/12/applications-of-cryptography/). *blogs.ucl.ac.uk*. Archived (https://web.archive.org/web/20180226185448/https://blogs.ucl.ac.uk/infosec/2017/03/12/applications-of-cryptography/) from the original on 26 February 2018. Retrieved 21 December 2021.
68. "Crypto FAQ: What is cryptocurrency?" (https://cryptocurrencyworks.com/faq//faq/what-is-cryptocurrency.html). *CryptoCurrency Works™*. Retrieved 27 August 2022.
69. "Crypto FAQ: What is cryptoeconomics & how does it work?" (https://cryptoeconworks.com/faq/what-is-cryptoeconomics.html). *cryptoeconworks.com*. Retrieved 27 August 2022.
70. "6.5.1 What Are the Cryptographic Policies of Some Countries?" (http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/cryptographic-policies-countries.htm). RSA Laboratories. Retrieved 26 March 2015.
71. Rosenoer, Jonathan (1995). "Cryptography & Speech". *CyberLaw*. "Archived copy" (https://web.archive.org/web/20051201184530/http://www.cyberlaw.com/cylw1095.html). Archived from the original (http://www.cyberlaw.com/cylw1095.html) on 1 December 2005. Retrieved 23 June 2006.
72. "Case Closed on Zimmermann PGP Investigation" (http://www.ieee-security.org/Cipher/Newsbriefs/1996/960214.zimmerman.html). *IEEE Computer Society's Technical Committee on Security and Privacy*. 14 February 1996. Retrieved 26 March 2015.
73. Levy, Steven (2001). *Crypto: How the Code Rebels Beat the Government—Saving Privacy in the Digital Age*. Penguin Books. p. 56. ISBN 978-0-14-024432-8. OCLC 244148644 (https://www.worldcat.org/oclc/244148644).
74. "Bernstein v USDOJ" (http://www.epic.org/crypto/export_controls/bernstein_decision_9_cir.html). *Electronic Privacy Information Center*. United States Court of Appeals for the Ninth Circuit. 6 May 1999. Retrieved 26 March 2015.
75. "Dual-use List – Category 5 – Part 2 – "Information Security" " (https://www.bis.doc.gov/index.php/documents/regulations-docs/445-category-5-part-2-information-security/file) (PDF). *Wassenaar Arrangement*. Retrieved 26 March 2015.
76. ".4 United States Cryptography Export/Import Laws" (http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/united-states-cryptography-export-import.htm). *RSA Laboratories*. Retrieved 26 March 2015.
77. Schneier, Bruce (15 June 2000). "The Data Encryption Standard (DES)" (http://www.schneier.com/crypto-gram-0006.html#DES). *Crypto-Gram*. Retrieved 26 March 2015.
78. Coppersmith, D. (May 1994). "The Data Encryption Standard (DES) and its strength against attacks" (http://domino.watson.ibm.com/tchjr/journalindex.nsf/0/94f78816c77fc77885256bfa0067fb98?OpenDocument) (PDF). *IBM Journal of Research and Development*. **38** (3): 243–250. doi:10.1147/rd.383.0243 (https://doi.org/10.1147%2Frd.383.0243). Retrieved 26 March 2015.
79. Biham, E.; Shamir, A. (1991). "Differential cryptanalysis of DES-like cryptosystems". *Journal of Cryptology*. **4** (1): 3–72. doi:10.1007/bf00630563 (https://doi.org/10.1007%2Fbf00630563). S2CID 206783462 (https://api.semanticscholar.org/CorpusID:206783462).
80. "The Digital Millennium Copyright Act of 1998" (http://www.copyright.gov/legislation/dmca.pdf) (PDF). *United States Copyright Office*. Retrieved 26 March 2015.
81. Ferguson, Niels (15 August 2001). "Censorship in action: why I don't publish my HDCP results" (https://web.archive.org/web/20011201184919/http://www.macfergus.com/niels/dmca/cia.html). Archived from the original (http://www.macfergus.com/niels/dmca/cia.html) on 1 December 2001. Retrieved 16 February 2009.
82. Schneier, Bruce (6 August 2001). "Arrest of Computer Researcher Is Arrest of First Amendment Rights" (https://www.schneier.com/essays/archives/2001/08/arrest_of_computer_r.html). InternetWeek. Retrieved 7 March 2017.
83. Williams, Christopher (11 August 2009). "Two convicted for refusal to decrypt data" (https://www.theregister.co.uk/2009/08/11/ripa_iii_figures/). *The Register*. Retrieved 26 March 2015.

84. Williams, Christopher (24 November 2009). "UK jails schizophrenic for refusal to decrypt files" (https://www.theregister.co.uk/2009/11/24/ripa_jfl/). *The Register*. Retrieved 26 March 2015.
85. Ingold, John (4 January 2012). "Password case reframes Fifth Amendment rights in context of digital world" (http://www.denverpost.com/news/ci_19669803). *The Denver Post*. Retrieved 26 March 2015.
86. Leyden, John (13 July 2011). "US court test for rights not to hand over crypto keys" (https://www.theregister.co.uk/2011/07/13/eff_piles_in_against_forced_decryption/). *The Register*. Retrieved 26 March 2015.
87. "Order Granting Application under the All Writs Act Requiring Defendant Fricosu to Assist in the Execution of Previously Issued Search Warrants" (https://www.wired.com/images_blogs/threatlevel/2012/01/decrypt.pdf) (PDF). United States District Court for the District of Colorado. Retrieved 26 March 2015.

Further reading

- Arbib, Jonathan; Dwyer, John (31 January 2011). *Discrete Mathematics for Cryptography* (1 ed.). Algana Publishing. ISBN 978-1-907934-01-8.
- Becket, B (1988). *Introduction to Cryptology*. Blackwell Scientific Publications. ISBN 978-0-632-01836-9. OCLC 16832704 (https://www.worldcat.org/oclc/16832704). Excellent coverage of many classical ciphers and cryptography concepts and of the "modern" DES and RSA systems.
- *Cryptography and Mathematics* by Bernhard Esslinger, 200 pages, part of the free open-source package CrypTool, "PDF download" (https://web.archive.org/web/20110722183013/http://www.cryptool.org/download/CrypToolScript-en.pdf) (PDF). Archived from the original on 22 July 2011. Retrieved 23 December 2013.. CrypTool is the most widespread e-learning program about cryptography and cryptanalysis, open source.
- *In Code: A Mathematical Journey* by Sarah Flannery (with David Flannery). Popular account of Sarah's award-winning project on public-key cryptography, co-written with her father.
- James Gannon, *Stealing Secrets, Telling Lies: How Spies and Codebreakers Helped Shape the Twentieth Century*, Washington, D.C., Brassey's, 2001, ISBN 1-57488-367-4.
- Oded Goldreich, *Foundations of Cryptography* (http://www.wisdom.weizmann.ac.il/~oded/foc-book.html), in two volumes, Cambridge University Press, 2001 and 2004.
- *Alvin's Secret Code* by Clifford B. Hicks (children's novel that introduces some basic cryptography and cryptanalysis).
- *Introduction to Modern Cryptography (http://www.cs.umd.edu/~jkatz/imc.html)* by Jonathan Katz and Yehuda Lindell.
- Ibrahim A. Al-Kadi, "The Origins of Cryptology: the Arab Contributions," Cryptologia, vol. 16, no. 2 (April 1992), pp. 97–126.
- Christof Paar (https://web.archive.org/web/20060709111152/http://www.crypto.rub.de/en_paar.html), Jan Pelzl, *Understanding Cryptography, A Textbook for Students and Practitioners*. (http://www.cryptography-textbook.com/) Archived (https://web.archive.org/web/20201031190651/http://cryptography-textbook.com/) 31 October 2020 at the Wayback Machine Springer, 2009. (Slides, online cryptography lectures and other information are available on the companion web site.) Very accessible introduction to practical cryptography for non-mathematicians.
- "Max Planck Encyclopedia of Public International Law" (http://www.mpepil.com)., giving an overview of international law issues regarding cryptography.
- *Introduction to Modern Cryptography* by Phillip Rogaway and Mihir Bellare, a mathematical introduction to theoretical cryptography including reduction-based security proofs. PDF download (http://www.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf).
- Stallings, William (March 2013). *Cryptography and Network Security: Principles and Practice* (6th ed.). Prentice Hall. ISBN 978-0-13-335469-0.

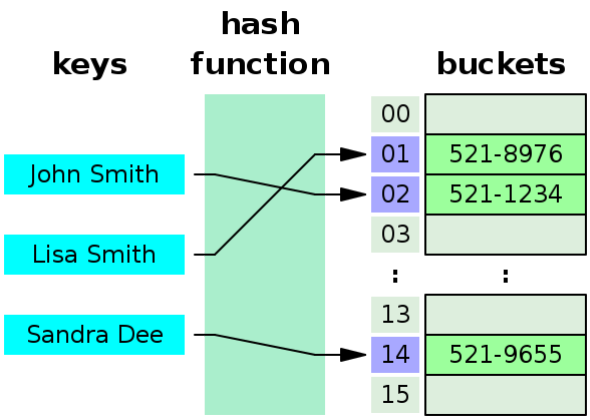
- Tenzer, Theo (2021): *SUPER SECRETO – The Third Epoch of Cryptography: Multiple, exponential, quantum-secure and above all, simple and practical Encryption for Everyone*, Norderstedt, ISBN 9783755761174.
- Johann-Christoph Woltag, 'Coded Communications (Encryption)' in Rüdiger Wolfrum (ed) *Max Planck Encyclopedia of Public International Law* (Oxford University Press 2009).

External links

- The dictionary definition of *cryptography* at Wiktionary
- Media related to Cryptography at Wikimedia Commons
- Cryptography (https://www.bbc.co.uk/programmes/p004y272) on *In Our Time* at the BBC
- Cryptography Works - web information portal (https://cryptographyworks.com)
- Crypto Glossary and Dictionary of Technical Cryptography (https://ciphersbyritter.com/GLOSSARY.HTM)
- A Course in Cryptography (https://www.cs.cornell.edu/courses/cs4830/2010fa/lecnotes.pdf) by Raphael Pass & Abhi Shelat – offered at Cornell in the form of lecture notes.
- For more on the use of cryptographic elements in fiction, see: Dooley, John F., William and Marilyn Ingersoll Professor of Computer Science, Knox College (23 August 2012). "Cryptology in Fiction" (https://web.archive.org/web/20200729044734/http://faculty.knox.edu/jdooley/Crypto/CryptoFiction.htm). Archived from the original (http://faculty.knox.edu/jdooley/Crypto/CryptoFiction.htm) on 29 July 2020. Retrieved 20 February 2015.
- The George Fabyan Collection (https://www.loc.gov/rr/rarebook/coll/073.html) at the Library of Congress has early editions of works of seventeenth-century English literature, publications relating to cryptography.

Data structure

In computer science, a **data structure** is a data organization, management, and storage format that is usually chosen for efficient access to data.[1][2][3] More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data,[4] i.e., it is an algebraic structure about data.



A data structure known as a hash table.

Contents

Usage

Data structures serve as the basis for abstract data types (ADT). The ADT defines the logical form of the data type. The data structure implements the physical form of the data type.[5]

Different types of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, relational databases commonly use B-tree indexes for data retrieval,[6] while compiler implementations usually use hash tables to look up identifiers.[7]

Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures can be used to organize the storage and retrieval of information stored in both main memory and secondary memory.[8]

Implementation

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer—a bit string, representing a memory address, that can be itself stored in memory and manipulated by the program. Thus, the array and record data

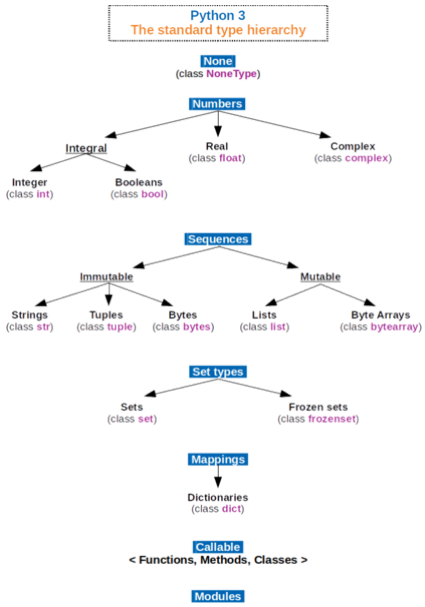
structures are based on computing the addresses of data items with arithmetic operations, while the linked data structures are based on storing addresses of data items within the structure itself.

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).[9]

Examples

There are numerous types of data structures, generally built upon simpler primitive data types. Well known examples are:[10]

- An *array* is a number of elements in a specific order, typically all of the same type (depending on the language, individual elements may either all be forced to be the same type, or may be of almost any type). Elements are accessed using an integer index to specify which element is required. Typical implementations allocate contiguous memory words for the elements of arrays (but this is not always a necessity). Arrays may be fixed-length or resizable.
- A *linked list* (also just called *list*) is a linear collection of data elements of any type, called nodes, where each node has itself a value, and points to the next node in the linked list. The principal advantage of a linked list over an array is that values can always be efficiently inserted and removed without relocating the rest of the list. Certain other operations, such as random access to a certain element, are however slower on lists than on arrays.



The standard type hierarchy of the programming language Python 3.

- A *record* (also called *tuple* or *struct*) is an aggregate data structure. A record is a value that contains other values, typically in fixed number and sequence and typically indexed by names. The elements of records are usually called *fields* or *members*. In the context of object-oriented programming, records are known as plain old data structures to distinguish them from objects.[11]
- *Hash tables*, *graphs* and *binary trees*.

Language support

Most assembly languages and some low-level languages, such as BCPL (Basic Combined Programming Language), lack built-in support for data structures. On the other hand, many high-level programming languages and some higher-level assembly languages, such as MASM, have special syntax or other built-in support for certain data structures, such as records and arrays. For example, the C (a direct descendant of BCPL) and Pascal languages support structs and records, respectively, in addition to vectors (one-dimensional arrays) and multi-dimensional arrays.[12][13]

Most programming languages feature some sort of library mechanism that allows data structure implementations to be reused by different programs. Modern languages usually come with standard libraries that implement the most common data structures. Examples are the C++ Standard Template Library, the Java Collections Framework, and the Microsoft .NET Framework.

Modern languages also generally support modular programming, the separation between the interface of a library module and its implementation. Some provide opaque data types that allow clients to hide implementation details. Object-oriented programming languages, such as C++, Java, and Smalltalk, typically use classes for this purpose.

Many known data structures have concurrent versions which allow multiple computing threads to access a single concrete instance of a data structure simultaneously.[14]

See also

- Abstract data type
- Concurrent data structure
- Data model
- Dynamization
- Linked data structure
- List of data structures
- Persistent data structure
- Plain old data structure
- Queap
- Succinct data structure
- Tree (data structure)

References

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). *Introduction to Algorithms, Third Edition* (https://dl.acm.org/citation.cfm?id=1614191) (3rd ed.). The MIT Press. ISBN 978-0262033848.
2. Black, Paul E. (15 December 2004). "data structure" (https://xlinux.nist.gov/dads/HTML/datastructur.html). In Pieterse, Vreda; Black, Paul E. (eds.). *Dictionary of Algorithms and Data Structures [online]*. National Institute of Standards and Technology. Retrieved 2018-11-06.
3. "Data structure" (https://www.britannica.com/technology/data-structure). *Encyclopaedia Britannica*. 17 April 2017. Retrieved 2018-11-06.
4. Wegner, Peter; Reilly, Edwin D. (2003-08-29). *Encyclopedia of Computer Science* (http://dl.acm.org/citation.cfm?id=1074100.1074312). Chichester, UK: John Wiley and Sons. pp. 507–512. ISBN 978-0470864128.
5. "Abstract Data Types" (https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/ADT.html). *Virginia Tech - CS3 Data Structures & Algorithms*.
6. Gavin Powell (2006). "Chapter 8: Building Fast-Performing Database Models" (http://searchsecurity.techtarget.com/generic/0,295582,sid87_gci1184450,00.html). *Beginning Database Design*. Wrox Publishing. ISBN 978-0-7645-7490-0.
7. "1.5 Applications of a Hash Table" (https://web.archive.org/web/20210427183057/https://www.cs.uregina.ca/Links/class-info/210/Hash/). *University of Regina - CS210 Lab: Hash Table*. Archived from the original (http://www.cs.uregina.ca/Links/class-info/210/Hash/) on 2021-04-27. Retrieved 2018-06-14.
8. "When data is too big to fit into the main memory" (http://homes.sice.indiana.edu/yye/lab/teaching/spring2014-C343/datatoobig.php). *homes.sice.indiana.edu*.
9. Dubey, R. C. (2014). *Advanced biotechnology : For B Sc and M Sc students of biotechnology and other biological sciences*. New Delhi: S Chand. ISBN 978-81-219-4290-4. OCLC 883695533 (https://www.worldcat.org/oclc/883695533).
10. Seymour, Lipschutz (2014). *Data structures* (Revised first ed.). New Delhi, India: McGraw Hill Education. ISBN 9781259029967. OCLC 927793728 (https://www.worldcat.org/oclc/927793728).
11. Walter E. Brown (September 29, 1999). "C++ Language Note: POD Types" (https://web.archive.org/web/20161203130543/http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/ISOcxx/doc/POD.html). Fermi National Accelerator Laboratory. Archived from the original (http://www.fnal.gov/docs/working-groups/fpcltf/Pkg/ISOcxx/doc/POD.html) on 2016-12-03. Retrieved 6 December 2016.

12. "The GNU C Manual" (https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html). Free Software Foundation. Retrieved 2014-10-15.
13. Van Canneyt, Michaël (September 2017). "Free Pascal: Reference Guide" (http://www.freepascal.org/docs-html/ref/ref.html). Free Pascal.
14. Mark Moir and Nir Shavit. "Concurrent Data Structures" (https://www.cs.tau.ac.il/~shanir/concurrent-data-structures.pdf) (PDF). *cs.tau.ac.il*.

Bibliography

- Peter Brass, *Advanced Data Structures*, Cambridge University Press, 2008, ISBN 978-0521880374
- Donald Knuth, *The Art of Computer Programming*, vol. 1. Addison-Wesley, 3rd edition, 1997, ISBN 978-0201896831
- Dinesh Mehta and Sartaj Sahni, *Handbook of Data Structures and Applications*, Chapman and Hall/CRC Press, 2004, ISBN 1584884355
- Niklaus Wirth, *Algorithms and Data Structures*, Prentice Hall, 1985, ISBN 978-0130220059

Further reading

- Alfred Aho, John Hopcroft, and Jeffrey Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983, ISBN 0-201-00023-7
- G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures - in Pascal and C (https://users.dcc.uchile.cl/~rbaeza/handbook/hbook.html)*, second edition, Addison-Wesley, 1991, ISBN 0-201-41607-7
- Ellis Horowitz and Sartaj Sahni, *Fundamentals of Data Structures in Pascal*, Computer Science Press, 1984, ISBN 0-914894-94-3

External links

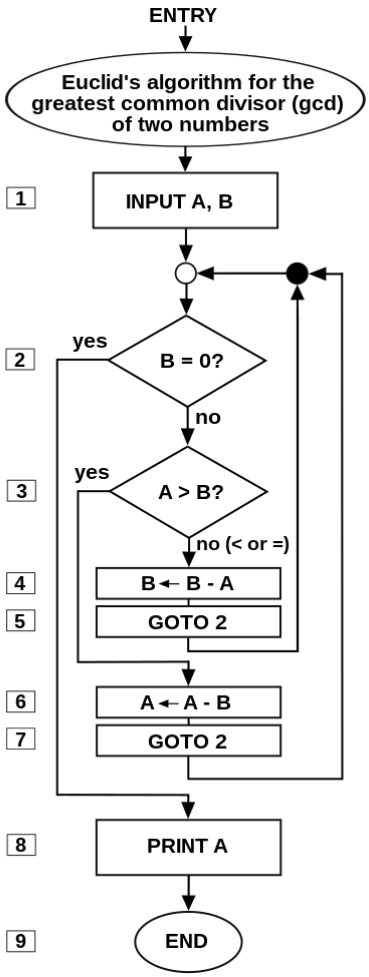
- Descriptions (https://web.archive.org/web/20050624234059/http://www.nist.gov/dads/) from the Dictionary of Algorithms and Data Structures
- Data structures course (http://www.cs.auckland.ac.nz/software/AlgAnim/ds_ToC.html)
- An Examination of Data Structures from .NET perspective (http://msdn.microsoft.com/en-us/library/aa289148(VS.71).aspx)
- Schaffer, C. *Data Structures and Algorithm Analysis* (http://people.cs.vt.edu/~shaffer/Book/C++3e20110915.pdf)

Algorithm

In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ (listen)) is a finite sequence of rigorous instructions, typically used to solve a class of specific problems or to perform a computation.[1] Algorithms are used as specifications for performing calculations and data processing. More advanced algorithms can perform automated deductions (referred to as automated reasoning) and use mathematical and logical tests to divert the code execution through various routes (referred to as automated decision-making). Using human characteristics as descriptors of machines in metaphorical ways was already practiced by Alan Turing with terms such as "memory", "search" and "stimulus".[2]

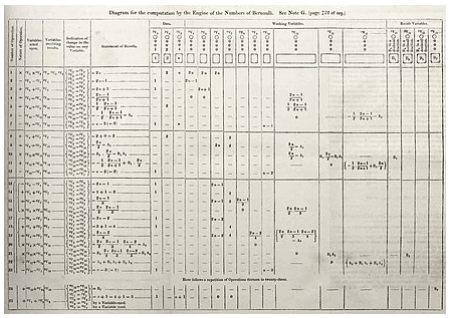
In contrast, a heuristic is an approach to problem solving that may not be fully specified or may not guarantee correct or optimal results, especially in problem domains where there is no well-defined correct or optimal result.[3]

As an effective method, an algorithm can be expressed within a finite amount of space and time,[4] and in a well-defined formal language[5] for calculating a function.[6] Starting from an initial state and initial input (perhaps empty),[7] the instructions describe a computation that, when executed, proceeds through a finite[8] number of well-defined successive states, eventually producing "output"[9] and terminating at a final ending state. The transition from one state to the next is not necessarily deterministic; some algorithms, known as randomized algorithms, incorporate random input.[10]



Flowchart of an algorithm (Euclid's algorithm) for calculating the greatest common divisor (g.c.d.) of two numbers *a* and *b* in locations named A and B. The algorithm proceeds by successive subtractions in two loops: IF the test B ≥ A yields "yes" or "true" (more accurately, the *number b* in location B is greater than or equal to the *number a* in location A) THEN, the algorithm specifies B ← B – A (meaning the number *b* – *a* replaces the old *b*). Similarly, IF A > B, THEN A ← A – B. The process terminates when (the contents of) B is 0, yielding the g.c.d. in A. (Algorithm derived from Scott 2009:13; symbols and drawing style from Tausworthe 1977).

Contents



Ada Lovelace's diagram from "note G", the first published computer algorithm

History

The concept of algorithm has existed since antiquity. Arithmetic algorithms, such as a division algorithm, were used by ancient Babylonian mathematicians c. 2500 BC and Egyptian mathematicians c. 1550 BC.[11] Greek mathematicians later used algorithms in 240 BC in the sieve of Eratosthenes for finding prime numbers, and the Euclidean algorithm for finding the greatest common divisor of two numbers.[12] Arabic mathematicians such as al-Kindi in the 9th century used cryptographic algorithms for code-breaking, based on frequency analysis.[13]

The word *algorithm* is derived from the name of the 9th-century Persian mathematician Muḥammad ibn Mūsā al-Khwārizmī, whose nisba (identifying him as from Khwarazm) was Latinized *as Algoritmi (Arabized Persian* الخوارزمی *c. 780–850).*[14][15] Muḥammad ibn Mūsā al-Khwārizmī was a mathematician, astronomer, geographer, and scholar in the House of Wisdom in Baghdad, whose name means 'the native of Khwarazm', a region that was part of Greater Iran and is now in Uzbekistan.[16][17] About 825, al-Khwarizmi wrote an Arabic language treatise on the Hindu–Arabic numeral system, which was translated into Latin during the 12th century. The

manuscript starts with the phrase *Dixit Algorizmi* ('Thus spake Al-Khwarizmi'), where "Algorizmi" was the translator's Latinization of Al-Khwarizmi's name.[18] Al-Khwarizmi was the most widely read mathematician in Europe in the late Middle Ages, primarily through another of his books, the Algebra.[19] In late medieval Latin, *algorismus*, English 'algorism', the corruption of his name, simply meant the "decimal number system".[20] In the 15th century, under the influence of the Greek word ἀριθμός (*arithmos*), 'number' (*cf.* 'arithmetic'), the Latin word was altered to *algorithmus*, and the corresponding English term 'algorithm' is first attested in the 17th century; the modern sense was introduced in the 19th century.[21]

Indian mathematics was predominantly algorithmic. Algorithms that are representative of the Indian mathematical tradition range from the ancient *Śulbasūtrās* to the medieval texts of the Kerala School.[22]

In English, the word *algorithm* was first used in about 1230 and then by Chaucer in 1391. English adopted the French term, but it was not until the late 19th century that "algorithm" took on the meaning that it has in modern English.[23]

Another early use of the word is from 1240, in a manual titled *Carmen de Algorismo* composed by Alexandre de Villedieu. It begins with:

> *Haec algorismus ars praesens dicitur, in qua / Talibus Indorum fruimur bis quinque figuris.*

which translates to:

> Algorism is the art by which at present we use those Indian figures, which number two times five.

The poem is a few hundred lines long and summarizes the art of calculating with the new styled Indian dice (*Tali Indorum*), or Hindu numerals.[24]

A partial formalization of the modern concept of algorithm began with attempts to solve the *Entscheidungsproblem* (decision problem) posed by David Hilbert in 1928. Later formalizations were framed as attempts to define "effective calculability"[25] or "effective method".[26] Those formalizations included the Gödel–Herbrand–Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's Formulation 1 of 1936, and Alan Turing's Turing machines of 1936–37 and 1939.

Informal definition

An informal definition could be "a set of rules that precisely defines a sequence of operations",[27] which would include all computer programs (including programs that do not perform numeric calculations), and (for example) any prescribed bureaucratic procedure[28] or cook-book recipe.[29]

In general, a program is only an algorithm if it stops eventually[30]—even though infinite loops may sometimes prove desirable.

A prototypical example of an algorithm is the Euclidean algorithm, which is used to determine the maximum common divisor of two integers; an example (there are others) is described by the flowchart above and as an example in a later section.

Boolos, Jeffrey & 1974, 1999 offer an informal meaning of the word "algorithm" in the following quotation:

> No human being can write fast enough, or long enough, or small enough† (†"smaller and smaller without limit ... you'd be trying to write on molecules, on atoms, on electrons") to list all members of an enumerably infinite set by writing out their names, one after another, in some notation. But humans can do something equally useful, in the case of certain enumerably infinite sets: They can give *explicit instructions for determining the **n**th member of the set*, for arbitrary finite *n*. Such instructions are to be given quite explicitly, in a form in which *they could be followed by a computing machine*, or by a *human who is capable of carrying out only very elementary operations on symbols*.[31]

An "enumerably infinite set" is one whose elements can be put into one-to-one correspondence with the integers. Thus Boolos and Jeffrey are saying that an algorithm implies instructions for a process that "creates" output integers from an *arbitrary* "input" integer or integers that, in theory, can be arbitrarily large. For example, an algorithm can be an algebraic equation such as $y = m + n$ (i.e., two arbitrary "input variables" m and n that produce an output y), but various authors' attempts to define the notion indicate that the word implies much more than this, something on the order of (for the addition example):

> Precise instructions (in a language understood by "the computer")[32] for a fast, efficient, "good"[33] process that specifies the "moves" of "the computer" (machine or human, equipped with the necessary internally contained information and capabilities)[34] to find, decode, and then process arbitrary input integers/symbols *m* and *n*, symbols *+* and *=* ... and "effectively"[35] produce, in a "reasonable" time,[36] output-integer *y* at a specified place and in a specified format.

The concept of *algorithm* is also used to define the notion of decidability—a notion that is central for explaining how formal systems come into being starting from a small set of axioms and rules. In logic, the time that an algorithm requires to complete cannot be measured, as it is not apparently related to the customary physical dimension. From such uncertainties, that characterize ongoing work, stems the unavailability of a definition of *algorithm* that suits both concrete (in some sense) and abstract usage of the term.

Most algorithms are intended to be implemented as computer programs. However, algorithms are also implemented by other means, such as in a biological neural network (for example, the human brain implementing arithmetic or an insect looking for food), in an electrical circuit, or in a mechanical device.

Formalization

Algorithms are essential to the way computers process data. Many computer programs contain algorithms that detail the specific instructions a computer should perform—in a specific order—to carry out a specified task, such as calculating employees' paychecks or printing students' report cards. Thus, an algorithm can be considered to be any sequence of operations that can be simulated by a Turing-complete system. Authors who assert this thesis include Minsky (1967), Savage (1987) and Gurevich (2000):

> Minsky: "But we will also maintain, with Turing ... that any procedure which could "naturally" be called effective, can, in fact, be realized by a (simple) machine. Although this may seem extreme, the arguments ... in its favor are hard to refute".[37] Gurevich:

> "... Turing's informal argument in favor of his thesis justifies a stronger thesis: every algorithm can be simulated by a Turing machine ... according to Savage [1987], an algorithm is a computational process defined by a Turing machine".[38]

Turing machines can define computational processes that do not terminate. The informal definitions of algorithms generally require that the algorithm always terminates. This requirement renders the task of deciding whether a formal procedure is an algorithm impossible in the general case—due to a major theorem of computability theory known as the halting problem.

Typically, when an algorithm is associated with processing information, data can be read from an input source, written to an output device and stored for further processing. Stored data are regarded as part of the internal state of the entity performing the algorithm. In practice, the state is stored in one or more data structures.

For some of these computational processes, the algorithm must be rigorously defined: specified in the way it applies in all possible circumstances that could arise. This means that any conditional steps must be systematically dealt with, case-by-case; the criteria for each case must be clear (and computable).

Because an algorithm is a precise list of precise steps, the order of computation is always crucial to the functioning of the algorithm. Instructions are usually assumed to be listed explicitly, and are described as starting "from the top" and going "down to the bottom"—an idea that is described more formally by *flow of control*.

So far, the discussion on the formalization of an algorithm has assumed the premises of imperative programming. This is the most common conception—one which attempts to describe a task in discrete, "mechanical" means. Unique to this conception of formalized algorithms is the assignment operation, which sets the value of a variable. It derives from the intuition of "memory" as a scratchpad. An example of such an assignment can be found below.

For some alternate conceptions of what constitutes an algorithm, see functional programming and logic programming.

Expressing algorithms

Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, drakon-charts, programming languages or control tables (processed by interpreters). Natural language expressions of algorithms tend to be verbose and ambiguous, and are rarely used for complex or technical algorithms. Pseudocode, flowcharts, drakon-charts and control tables are structured ways to express algorithms that avoid many of the ambiguities common in the statements based on natural language. Programming languages are primarily intended for expressing algorithms in a form that can be executed by a computer, but are also often used as a way to define or document algorithms.

There is a wide variety of representations possible and one can express a given Turing machine program as a sequence of machine tables (see finite-state machine, state transition table and control table for more), as flowcharts and drakon-charts (see state diagram for more), or as a form of rudimentary machine code or assembly code called "sets of quadruples" (see Turing machine for more).

Representations of algorithms can be classed into three accepted levels of Turing machine description, as follows:[39]

1 High-level description

"...prose to describe an algorithm, ignoring the implementation details. At this level, we do not need to mention how the machine manages its tape or head."

2 Implementation description

"...prose used to define the way the Turing machine uses its head and the way that it stores data on its tape. At this level, we do not give details of states or transition function."

3 Formal description

Most detailed, "lowest level", gives the Turing machine's "state table".

For an example of the simple algorithm "Add m+n" described in all three levels, see Examples.

Design

Algorithm design refers to a method or a mathematical process for problem-solving and engineering algorithms. The design of algorithms is part of many solution theories of operation research, such as dynamic programming and divide-and-conquer. Techniques for designing and implementing algorithm designs are also called algorithm design patterns,[40] with examples including the template method pattern and the decorator pattern.

One of the most important aspects of algorithm design is resource (run-time, memory usage) efficiency; the big O notation is used to describe e.g. an algorithm's run-time growth as the size of its input increases.

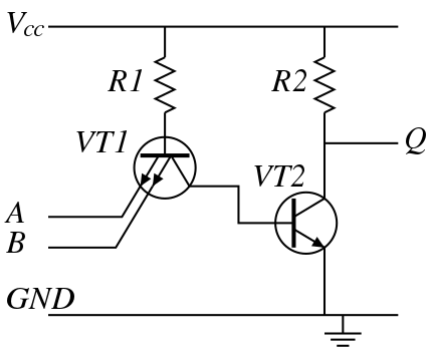
Typical steps in the development of algorithms:

1. Problem definition
2. Development of a model
3. Specification of the algorithm
4. Designing an algorithm
5. Checking the correctness of the algorithm
6. Analysis of algorithm
7. Implementation of algorithm
8. Program testing
9. Documentation preparation

Computer algorithms

"Elegant" (compact) programs, "good" (fast) programs : The notion of "simplicity and elegance" appears informally in Knuth and precisely in Chaitin:

> Knuth: " ... we want *good* algorithms in some loosely defined aesthetic sense. One criterion ... is the length of time taken to perform the algorithm .... Other criteria are adaptability of the algorithm to computers, its simplicity and elegance, etc."[41]
>
> Chaitin: " ... a program is 'elegant,' by which I mean that it's the smallest possible program for producing the output that it does"[42]



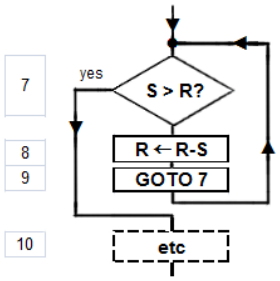
Logical NAND algorithm implemented electronically in 7400 chip

Chaitin prefaces his definition with: "I'll show you can't prove that a program is 'elegant' "—such a proof would solve the Halting problem (ibid).

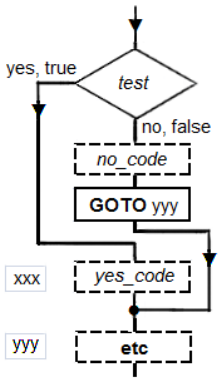
Algorithm versus function computable by an algorithm: For a given function multiple algorithms may exist. This is true, even without expanding the available instruction set available to the programmer. Rogers observes that "It is ... important to distinguish between the notion of *algorithm*, i.e. procedure and the notion of *function computable by algorithm*, i.e. mapping yielded by procedure. The same function may have several different algorithms".[43]

Unfortunately, there may be a tradeoff between goodness (speed) and elegance (compactness)—an elegant program may take more steps to complete a computation than one less elegant. An example that uses Euclid's algorithm appears below.

Computers (and computors), models of computation: A computer (or human "computor"[44]) is a restricted type of machine, a "discrete deterministic mechanical device"[45] that blindly follows its instructions.[46] Melzak's and Lambek's primitive models[47] reduced this notion to four elements: (i) discrete, distinguishable *locations*, (ii) discrete, indistinguishable *counters*[48] (iii) an agent, and (iv) a list of instructions that are *effective* relative to the capability of the agent.[49]



WHILE S ≤ R **DO** R ← R - S



IF test=true **THEN** *yes_code* **ELSE** no_code

Flowchart examples of the canonical Böhm-Jacopini structures: the SEQUENCE (rectangles descending the page), the WHILE-DO and the IF-THEN-ELSE. The three structures are made of the primitive conditional GOTO (`IF test THEN GOTO step xxx`, shown as diamond), the unconditional GOTO (rectangle), various assignment operators (rectangle), and HALT (rectangle). Nesting of these structures inside assignment-blocks result in complex diagrams (cf. Tausworthe 1977:100, 114).

Minsky describes a more congenial variation of Lambek's "abacus" model in his "Very Simple Bases for Computability".[50] Minsky's machine proceeds sequentially through its five (or six, depending on how one counts) instructions unless either a conditional IF-THEN GOTO or an unconditional GOTO changes program flow out of sequence. Besides HALT, Minsky's machine includes three *assignment* (replacement, substitution)[51] operations: ZERO (e.g. the contents of location replaced by 0: L ← 0), SUCCESSOR (e.g. L ← L+1), and DECREMENT (e.g. L ← L − 1).[52] Rarely must a programmer write "code" with such a limited instruction set. But Minsky shows (as do Melzak and Lambek) that his machine is Turing complete with only four general *types* of instructions: conditional GOTO, unconditional GOTO, assignment/replacement/substitution, and HALT. However, a few different assignment instructions (e.g. DECREMENT, INCREMENT, and ZERO/CLEAR/EMPTY for a Minsky machine) are also required for Turing-completeness; their exact specification is somewhat up to the designer. The unconditional GOTO is a convenience; it can be constructed by initializing a dedicated location to zero e.g. the instruction " Z ← 0 "; thereafter the instruction IF Z=0 THEN GOTO xxx is unconditional.

Simulation of an algorithm: computer (computor) language: Knuth advises the reader that "the best way to learn an algorithm is to try it . . . immediately take pen and paper and work through an example".[53] But what about a simulation or execution of the real thing? The programmer must translate the algorithm into a language that the simulator/computer/computor can *effectively* execute. Stone gives an example of this: when computing the roots of a quadratic equation the computor must know how to take a square root. If they don't, then the algorithm, to be effective, must provide a set of rules for extracting a square root.[54]

This means that the programmer must know a "language" that is effective relative to the target computing agent (computer/computor).

But what model should be used for the simulation? Van Emde Boas observes "even if we base complexity theory on abstract instead of concrete machines, arbitrariness of the choice of a model remains. It is at this point that the notion of *simulation* enters".[55] When speed is being measured, the instruction set matters. For example, the subprogram in Euclid's algorithm to compute the remainder would execute much faster if the programmer had a "modulus" instruction available rather than just subtraction (or worse: just Minsky's "decrement").

Structured programming, canonical structures: Per the Church–Turing thesis, any algorithm can be computed by a model known to be Turing complete, and per Minsky's demonstrations, Turing completeness requires only four instruction types—conditional GOTO, unconditional GOTO, assignment, HALT. Kemeny and Kurtz observe that, while "undisciplined" use of unconditional GOTOs and conditional IF-THEN GOTOs can result in "spaghetti code", a programmer can write structured programs using only these instructions; on the other hand "it is also possible, and not too hard, to write badly structured programs in a structured language".[56] Tausworthe augments the three Böhm-Jacopini canonical structures:[57] SEQUENCE, IF-THEN-ELSE, and WHILE-DO, with two more: DO-WHILE and CASE.[58] An additional benefit of a structured program is that it lends itself to proofs of correctness using mathematical induction.[59]

Canonical flowchart symbols[60]: The graphical aide called a flowchart, offers a way to describe and document an algorithm (and a computer program of one). Like the program flow of a Minsky machine, a flowchart always starts at the top of a page and proceeds down. Its primary symbols are only four: the directed arrow showing program flow, the rectangle (SEQUENCE, GOTO), the diamond (IF-THEN-ELSE), and the dot (OR-tie). The Böhm–Jacopini canonical structures are made of these primitive shapes. Sub-structures can "nest" in rectangles, but only if a single exit occurs from the superstructure. The symbols, and their use to build the canonical structures are shown in the diagram.

Examples

Algorithm example

One of the simplest algorithms is to find the largest number in a list of numbers of random order. Finding the solution requires looking at every number in the list. From this follows a simple algorithm, which can be stated in a high-level description in English prose, as:

High-level description:

1. If there are no numbers in the set then there is no highest number.
2. Assume the first number in the set is the largest number in the set.
3. For each remaining number in the set: if this number is larger than the current largest number, consider this number to be the largest number in the set.
4. When there are no numbers left in the set to iterate over, consider the current largest number to be the largest number of the set.

(Quasi-)formal description: Written in prose but much closer to the high-level language of a computer program, the following is the more formal coding of the algorithm in pseudocode or pidgin code:

```
Algorithm LargestNumber
Input: A list of numbers L.
Output: The largest number in the list L.
```

```
if L.size = 0 return null
largest ← L[0]
for each item in L, do
if item > largest, then
largest ← item
return largest
```

- "←" denotes assignment. For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the following value.

Euclid's algorithm

In mathematics, the **Euclidean algorithm**, or **Euclid's algorithm**, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his *Elements* (c. 300 BC).[61] It is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

Euclid poses the problem thus: "Given two numbers not prime to one another, to find their greatest common measure". He defines "A number [to be] a multitude composed of units": a counting number, a positive integer not including zero. To "measure" is to place a shorter measuring length *s* successively (*q* times) along longer length *l* until the remaining portion *r* is less than the shorter length *s*.[62] In modern words, remainder $r = l - q \times s$, *q* being the quotient, or remainder *r* is the "modulus", the integer-fractional part left over after the division.[63]

For Euclid's method to succeed, the starting lengths must satisfy two requirements: (i) the lengths must not be zero, AND (ii) the subtraction must be "proper"; i.e., a test must guarantee that the smaller of the two numbers is subtracted from the larger (or the two can be equal so their subtraction yields zero).

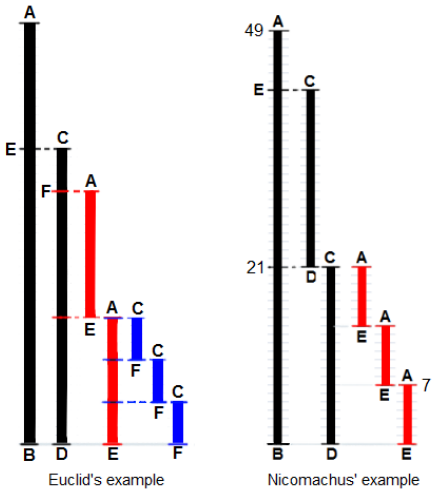
Euclid's original proof adds a third requirement: the two lengths must not be prime to one another. Euclid stipulated this so that he could construct a reductio ad absurdum proof that the two numbers' common measure is in fact the *greatest*.[64] While Nicomachus' algorithm is the same as Euclid's, when the numbers are prime to one another, it yields the number "1" for their common measure. So, to be precise, the following is really Nicomachus' algorithm.

Computer language for Euclid's algorithm

Only a few instruction *types* are required to execute Euclid's algorithm—some logical tests (conditional GOTO), unconditional GOTO, assignment (replacement), and subtraction.

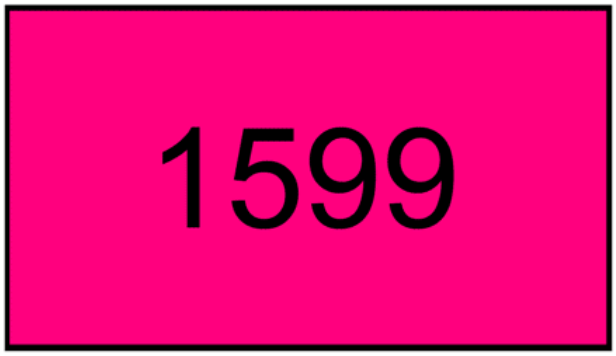
- A *location* is symbolized by upper case letter(s), e.g. S, A, etc.
- The varying quantity (number) in a location is written in lower case letter(s) and (usually) associated with the location's name. For example, location L at the start might contain the number *l* = 3009.

An



Euclid's example Nicomachus' example

The example-diagram of Euclid's algorithm from T.L. Heath (1908), with more detail added. Euclid does not go beyond a third measuring and gives no numerical examples. Nicomachus gives the example of 49 and 21: "I subtract the less from the greater; 28 is left; then again I subtract from this the same 21 (for this is possible); 7 is left; I subtract this from 21, 14 is left; from which I again subtract 7 (for this is possible); 7 is left, but 7 cannot be subtracted from 7." Heath comments that "The last phrase is curious, but the meaning of it is obvious enough, as also the meaning of the phrase about ending 'at one and the same number'."(Heath 1908:300).



A graphical expression of Euclid's algorithm to find the greatest common divisor for 1599 and 650.

```
1599 = 650×2 + 299
650 = 299×2 + 52
299 = 52×5 + 39
52 = 39×1 + 13
39 = 13×3 + 0
```

inelegant program for Euclid's algorithm

The following algorithm is framed as Knuth's four-step version of Euclid's and Nicomachus', but, rather than using division to find the remainder, it uses successive subtractions of the shorter length *s* from the remaining length *r* until *r* is less than *s*. The high-level description, shown in boldface, is adapted from Knuth 1973:2–4:

INPUT:

```
1 [Into two locations L and S put the numbers l and s that
represent the two lengths]:
INPUT L, S
2 [Initialize R: make the remaining length r equal to the
starting/initial/input length l]:
R ← L
```

E0: [Ensure *r* ≥ *s*.]

```
3 [Ensure the smaller of the two numbers is in S and the larger in R]:
IF R > S THEN
the contents of L is the larger number so skip over the exchange-steps 4, 5 and 6:
GOTO step 7
ELSE
swap the contents of R and S.
4 L ← R (this first step is redundant, but is useful for later discussion).
5 R ← S
6 S ← L
```

E1: [Find remainder]: Until the remaining length *r* in R is less than the shorter length *s* in S, repeatedly subtract the measuring number *s* in S from the remaining length *r* in R.

```
7 IF S > R THEN
done measuring so
GOTO 10
ELSE
```

```
measure again,
8 R ← R − S
9 [Remainder-loop]:
GOTO 7.
```

E2: [Is the remainder zero?]: EITHER (i) the last measure was exact, the remainder in R is zero, and the program can halt, OR (ii) the algorithm must continue: the last measure left a remainder in R less than measuring number in S.

```
10 IF R = 0 THEN
done so
GOTO step 15
ELSE
CONTINUE TO step 11,
```

E3: [Interchange *s* and *r*]: The nut of Euclid's algorithm. Use remainder *r* to measure what was previously smaller number *s*; L serves as a temporary location.

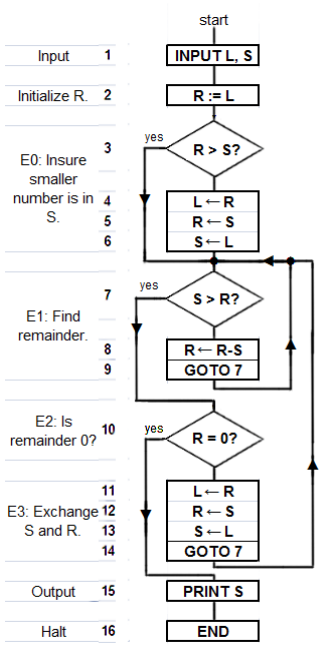
```
11 L ← R
12 R ← S
13 S ← L
14 [Repeat the measuring process]:
GOTO 7
```

OUTPUT:

```
15 [Done. S contains the greatest common divisor]:
PRINT S
```

DONE:

```
16 HALT, END, STOP.
```



"Inelegant"

"Inelegant" is a translation of Knuth's version of the algorithm with a subtraction-based remainder-loop replacing his use of division (or a "modulus" instruction). Derived from Knuth 1973:2–4. Depending on the two numbers "Inelegant" may compute the g.c.d. in fewer steps than "Elegant".

An elegant program for Euclid's algorithm

The following version of Euclid's algorithm requires only six core instructions to do what thirteen are required to do by "Inelegant"; worse, "Inelegant" requires more *types* of instructions. The flowchart of "Elegant" can be found at the top of this article. In the (unstructured) Basic language, the steps are numbered, and the instruction **LET** `[]` = `[]` is the assignment instruction symbolized by ←.

```
5 REM Euclid's algorithm for greatest common divisor
6 PRINT "Type two integers greater than 0"
10 INPUT A,B
20 IF B=0 THEN GOTO 80
30 IF A > B THEN GOTO 60
40 LET B=B-A
50 GOTO 20
60 LET A=A-B
70 GOTO 20
80 PRINT A
90 END
```

How "Elegant" works: In place of an outer "Euclid loop", "Elegant" shifts back and forth between two "co-loops", an A > B loop that computes A ← A − B, and a B ≤ A loop that computes B ← B − A. This works because, when at last the minuend M is less than or equal to the subtrahend S (Difference = Minuend − Subtrahend), the minuend can become *s* (the new measuring length) and the subtrahend can become the new *r* (the length to be measured); in other words the "sense" of the subtraction reverses.

The following version can be used with programming languages from the C-family:

```
// Euclid's algorithm for greatest common divisor
int euclidAlgorithm (int A, int B){
     A=abs(A);
     B=abs(B);
     while (B!=0){
          while (A>B) A=A-B;
          B=B-A;
     }
     return A;
}
```

Testing the Euclid algorithms

Does an algorithm do what its author wants it to do? A few test cases usually give some confidence in the core functionality. But tests are not enough. For test cases, one source[65] uses 3009 and 884. Knuth suggested 40902, 24140. Another interesting case is the two relatively prime numbers 14157 and 5950.

But "exceptional cases"[66] must be identified and tested. Will "Inelegant" perform properly when R > S, S > R, R = S? Ditto for "Elegant": B > A, A > B, A = B? (Yes to all). What happens when one number is zero, both numbers are zero? ("Inelegant" computes forever in all cases; "Elegant" computes forever when A = 0.) What happens if *negative* numbers are entered? Fractional numbers? If the input numbers, i.e. the domain of the function computed by the algorithm/program, is to include only positive integers including zero, then the failures at zero indicate that the algorithm (and the program that instantiates it) is a partial function rather than a total function. A notable failure due to exceptions is the Ariane 5 Flight 501 rocket failure (June 4, 1996).

Proof of program correctness by use of mathematical induction: Knuth demonstrates the application of mathematical induction to an "extended" version of Euclid's algorithm, and he proposes "a general method applicable to proving the validity of any algorithm".[67] Tausworthe proposes that a measure of the complexity of a program be the length of its correctness proof.[68]

Measuring and improving the Euclid algorithms

Elegance (compactness) versus goodness (speed): With only six core instructions, "Elegant" is the clear winner, compared to "Inelegant" at thirteen instructions. However, "Inelegant" is *faster* (it arrives at HALT in fewer steps). Algorithm analysis[69] indicates why this is the case: "Elegant" does *two* conditional tests in every subtraction loop, whereas "Inelegant" only does one. As the algorithm (usually) requires many loop-throughs, *on average* much time is wasted doing a "B = 0?" test that is needed only after the remainder is computed.

Can the algorithms be improved?: Once the programmer judges a program "fit" and "effective"—that is, it computes the function intended by its author—then the question becomes, can it be improved?

The compactness of "Inelegant" can be improved by the elimination of five steps. But Chaitin proved that compacting an algorithm cannot be automated by a generalized algorithm;[70] rather, it can only be done heuristically; i.e., by exhaustive search (examples to be found at Busy beaver), trial and error, cleverness, insight, application of inductive reasoning, etc. Observe that steps 4, 5 and 6 are repeated in steps 11, 12 and 13. Comparison with "Elegant" provides a hint that these steps, together with steps 2 and 3, can be eliminated. This reduces the number of core instructions from thirteen to eight, which makes it "more elegant" than "Elegant", at nine steps.

The speed of "Elegant" can be improved by moving the "B=0?" test outside of the two subtraction loops. This change calls for the addition of three instructions (B = 0?, A = 0?, GOTO). Now "Elegant" computes the example-numbers faster; whether this is always the case for any given A, B, and R, S would require a detailed analysis.

Algorithmic analysis

It is frequently important to know how much of a particular resource (such as time or storage) is theoretically required for a given algorithm. Methods have been developed for the analysis of algorithms to obtain such quantitative answers (estimates); for example, an algorithm which adds up the elements of a list of n numbers would have a time requirement of $O(n)$, using big O notation. At all times the algorithm only needs to remember two values: the sum of all the elements so far, and its current position in the input list. Therefore, it is said to have a space requirement of $O(1)$, if the space required to store the input numbers is not counted, or $O(n)$ if it is counted.

Different algorithms may complete the same task with a different set of instructions in less or more time, space, or 'effort' than others. For example, a binary search algorithm (with cost $O(\log n)$) outperforms a sequential search (cost $O(n)$) when used for table lookups on sorted lists or arrays.

Formal versus empirical

The analysis, and study of algorithms is a discipline of computer science, and is often practiced abstractly without the use of a specific programming language or implementation. In this sense, algorithm analysis resembles other mathematical disciplines in that it focuses on the underlying properties of the algorithm and not on the specifics of any particular implementation. Usually pseudocode is used for analysis as it is the simplest and most general representation. However, ultimately, most algorithms are usually implemented on particular hardware/software platforms and their algorithmic efficiency is eventually put to the test using real code. For the solution of a "one off" problem, the efficiency of a particular algorithm may not have significant consequences (unless n is extremely large) but for algorithms designed for fast interactive, commercial or long life scientific usage it may be critical. Scaling from small n to large n frequently exposes inefficient algorithms that are otherwise benign.

Empirical testing is useful because it may uncover unexpected interactions that affect performance. Benchmarks may be used to compare before/after potential improvements to an algorithm after program optimization. Empirical tests cannot replace formal analysis, though, and are not trivial to perform in a fair manner.[71]

Execution efficiency

To illustrate the potential improvements possible even in well-established algorithms, a recent significant innovation, relating to FFT algorithms (used heavily in the field of image processing), can decrease processing time up to 1,000 times for applications like medical imaging.[72] In general, speed improvements depend on special properties of the problem, which are very common

in practical applications.[73] Speedups of this magnitude enable computing devices that make extensive use of image processing (like digital cameras and medical equipment) to consume less power.

Classification

There are various ways to classify algorithms, each with its own merits.

By implementation

One way to classify algorithms is by implementation means.

Recursion

A recursive algorithm is one that invokes (makes reference to) itself repeatedly until a certain condition (also known as termination condition) matches, which is a method common to functional programming. Iterative algorithms use repetitive constructs like loops and sometimes additional data structures like stacks to solve the given problems. Some problems are naturally suited for one implementation or the other. For example, towers of Hanoi is well understood using recursive implementation. Every recursive version has an equivalent (but possibly more or less complex) iterative version, and vice versa.

```
int gcd(int A, int B) {
    if (B == 0)
        return A;
    else if (A > B)
        return gcd(A-B,B);
    else
        return gcd(A,B-A);
}
```

Recursive C implementation of Euclid's algorithm from the above flowchart

Logical

An algorithm may be viewed as controlled logical deduction. This notion may be expressed as: *Algorithm = logic + control*.[74] The logic component expresses the axioms that may be used in the computation and the control component determines the way in which deduction is applied to the axioms. This is the basis for the logic programming paradigm. In pure logic programming languages, the control component is fixed and algorithms are specified by supplying only the logic component. The appeal of this approach is the elegant semantics: a change in the axioms produces a well-defined change in the algorithm.

Serial, parallel or distributed

Algorithms are usually discussed with the assumption that computers execute one instruction of an algorithm at a time. Those computers are sometimes called serial computers. An algorithm designed for such an environment is called a serial algorithm, as opposed to parallel algorithms or distributed algorithms. Parallel algorithms take advantage of computer architectures where several processors can work on a problem at the same time, whereas distributed algorithms utilize multiple machines connected with a computer network. Parallel or distributed algorithms divide the problem into more symmetrical or asymmetrical subproblems and collect the results back together. The resource consumption in such algorithms is not only processor cycles on each processor but also the communication overhead between the processors. Some sorting algorithms can be parallelized efficiently, but their communication overhead is expensive. Iterative algorithms are generally parallelizable. Some problems have no parallel algorithms and are called inherently serial problems.

Deterministic or non-deterministic

Deterministic algorithms solve the problem with exact decision at every step of the algorithm whereas non-deterministic algorithms solve problems via guessing although typical guesses are made more accurate through the use of heuristics.

Exact or approximate

While many algorithms reach an exact solution, approximation algorithms seek an approximation that is closer to the true solution. The approximation can be reached by either using a deterministic or a random strategy. Such algorithms have practical value for many

hard problems. One of the examples of an approximate algorithm is the Knapsack problem, where there is a set of given items. Its goal is to pack the knapsack to get the maximum total value. Each item has some weight and some value. Total weight that can be carried is no more than some fixed number X. So, the solution must consider weights of items as well as their value.[75]

Quantum algorithm

They run on a realistic model of quantum computation. The term is usually used for those algorithms which seem inherently quantum, or use some essential feature of Quantum computing such as quantum superposition or quantum entanglement.

By design paradigm

Another way of classifying algorithms is by their design methodology or paradigm. There is a certain number of paradigms, each different from the other. Furthermore, each of these categories includes many different types of algorithms. Some common paradigms are:

Brute-force or exhaustive search

This is the naive method of trying every possible solution to see which is best.[76]

Divide and conquer

A divide-and-conquer algorithm repeatedly reduces an instance of a problem to one or more smaller instances of the same problem (usually recursively) until the instances are small enough to solve easily. One such example of divide and conquer is merge sorting. Sorting can be done on each segment of data after dividing data into segments and sorting of entire data can be obtained in the conquer phase by merging the segments. A simpler variant of divide and conquer is called a *decrease-and-conquer algorithm*, which solves an identical subproblem and uses the solution of this subproblem to solve the bigger problem. Divide and conquer divides the problem into multiple subproblems and so the conquer stage is more complex than decrease and conquer algorithms. An example of a decrease and conquer algorithm is the binary search algorithm.

Search and enumeration

Many problems (such as playing chess) can be modeled as problems on graphs. A graph exploration algorithm specifies rules for moving around a graph and is useful for such problems. This category also includes search algorithms, branch and bound enumeration and backtracking.

Randomized algorithm

Such algorithms make some choices randomly (or pseudo-randomly). They can be very useful in finding approximate solutions for problems where finding exact solutions can be impractical (see heuristic method below). For some of these problems, it is known that the fastest approximations must involve some randomness.[77] Whether randomized algorithms with polynomial time complexity can be the fastest algorithms for some problems is an open question known as the P versus NP problem. There are two large classes of such algorithms:

1. Monte Carlo algorithms return a correct answer with high-probability. E.g. RP is the subclass of these that run in polynomial time.
2. Las Vegas algorithms always return the correct answer, but their running time is only probabilistically bound, e.g. ZPP.

Reduction of complexity

This technique involves solving a difficult problem by transforming it into a better-known problem for which we have (hopefully) asymptotically optimal algorithms. The goal is to find a reducing algorithm whose complexity is not dominated by the resulting reduced algorithm's. For example, one selection algorithm for finding the median in an unsorted list involves first sorting the list (the expensive portion) and then pulling out the middle element in the sorted list (the cheap portion). This technique is also known as *transform and conquer*.

Back tracking

In this approach, multiple solutions are built incrementally and abandoned when it is determined that they cannot lead to a valid full solution.

Optimization problems

For optimization problems there is a more specific classification of algorithms; an algorithm for such problems may fall into one or more of the general categories described above as well as into one of the following:

Linear programming

When searching for optimal solutions to a linear function bound to linear equality and inequality constraints, the constraints of the problem can be used directly in producing the optimal solutions. There are algorithms that can solve any problem in this category, such as the popular simplex algorithm.[78] Problems that can be solved with linear programming include the maximum flow problem for directed graphs. If a problem additionally requires that one or more of the unknowns must be an integer then it is classified in integer programming. A linear programming algorithm can solve such a problem if it can be proved that all restrictions for integer values are superficial, i.e., the solutions satisfy these restrictions anyway. In the general case, a specialized algorithm or an algorithm that finds approximate solutions is used, depending on the difficulty of the problem.

Dynamic programming

When a problem shows optimal substructures—meaning the optimal solution to a problem can be constructed from optimal solutions to subproblems—and overlapping subproblems, meaning the same subproblems are used to solve many different problem instances, a quicker approach called *dynamic programming* avoids recomputing solutions that have already been computed. For example, Floyd–Warshall algorithm, the shortest path to a goal from a vertex in a weighted graph can be found by using the shortest path to the goal from all adjacent vertices. Dynamic programming and memoization go together. The main difference between dynamic programming and divide and conquer is that subproblems are more or less independent in divide and conquer, whereas subproblems overlap in dynamic programming. The difference between dynamic programming and straightforward recursion is in caching or memoization of recursive calls. When subproblems are independent and there is no repetition, memoization does not help; hence dynamic programming is not a solution for all complex problems. By using memoization or maintaining a table of subproblems already solved, dynamic programming reduces the exponential nature of many problems to polynomial complexity.

The greedy method

A greedy algorithm is similar to a dynamic programming algorithm in that it works by examining substructures, in this case not of the problem but of a given solution. Such algorithms start with some solution, which may be given or have been constructed in some way, and improve it by making small modifications. For some problems they can find the optimal solution while for others they stop at local optima, that is, at solutions that cannot be improved by the algorithm but are not optimum. The most popular use of greedy algorithms is for finding the minimal spanning tree where finding the optimal solution is possible with this method. Huffman Tree, Kruskal, Prim, Sollin are greedy algorithms that can solve this optimization problem.

The heuristic method

In optimization problems, heuristic algorithms can be used to find a solution close to the optimal solution in cases where finding the optimal solution is impractical. These algorithms work by getting closer and closer to the optimal solution as they progress. In principle, if run for an infinite amount of time, they will find the optimal solution. Their merit is that they can find a solution very close to the optimal solution in a relatively short time. Such algorithms include local search, tabu search, simulated annealing, and genetic algorithms. Some of them, like simulated annealing, are non-deterministic algorithms while others, like tabu search, are deterministic. When a bound on the error of the non-optimal solution is known, the algorithm is further categorized as an approximation algorithm.

By field of study

Every field of science has its own problems and needs efficient algorithms. Related problems in one field are often studied together. Some example classes are search algorithms, sorting algorithms, merge algorithms, numerical algorithms, graph algorithms, string algorithms, computational geometric algorithms, combinatorial algorithms, medical algorithms, machine learning, cryptography, data compression algorithms and parsing techniques.

Fields tend to overlap with each other, and algorithm advances in one field may improve those of other, sometimes completely unrelated, fields. For example, dynamic programming was invented for optimization of resource consumption in industry but is now used in solving a broad range of problems in many fields.

By complexity

Algorithms can be classified by the amount of time they need to complete compared to their input size:

- Constant time: if the time needed by the algorithm is the same, regardless of the input size. E.g. an access to an array element.
- Logarithmic time: if the time is a logarithmic function of the input size. E.g. binary search algorithm.
- Linear time: if the time is proportional to the input size. E.g. the traverse of a list.
- Polynomial time: if the time is a power of the input size. E.g. the bubble sort algorithm has quadratic time complexity.
- Exponential time: if the time is an exponential function of the input size. E.g. Brute-force search.

Some problems may have multiple algorithms of differing complexity, while other problems might have no algorithms or no known efficient algorithms. There are also mappings from some problems to other problems. Owing to this, it was found to be more suitable to classify the problems themselves instead of the algorithms into equivalence classes based on the complexity of the best possible algorithms for them.

Continuous algorithms

The adjective "continuous" when applied to the word "algorithm" can mean:

- An algorithm operating on data that represents continuous quantities, even though this data is represented by discrete approximations—such algorithms are studied in numerical analysis; or
- An algorithm in the form of a differential equation that operates continuously on the data, running on an analog computer.[79]

Legal issues

Algorithms, by themselves, are not usually patentable. In the United States, a claim consisting solely of simple manipulations of abstract concepts, numbers, or signals does not constitute "processes" (USPTO 2006), and hence algorithms are not patentable (as in Gottschalk v. Benson). However practical applications of algorithms are sometimes patentable. For example, in Diamond v. Diehr, the application of a simple feedback algorithm to aid in the curing of synthetic rubber was

deemed patentable. The patenting of software is highly controversial, and there are highly criticized patents involving algorithms, especially data compression algorithms, such as Unisys' LZW patent.

Additionally, some cryptographic algorithms have export restrictions (see export of cryptography).

History: Development of the notion of "algorithm"

Ancient Near East

The earliest evidence of algorithms is found in the Babylonian mathematics of ancient Mesopotamia (modern Iraq). A Sumerian clay tablet found in Shuruppak near Baghdad and dated to circa 2500 BC described the earliest division algorithm.[11] During the Hammurabi dynasty circa 1800-1600 BC, Babylonian clay tablets described algorithms for computing formulas.[80] Algorithms were also used in Babylonian astronomy. Babylonian clay tablets describe and employ algorithmic procedures to compute the time and place of significant astronomical events.[81]

Algorithms for arithmetic are also found in ancient Egyptian mathematics, dating back to the Rhind Mathematical Papyrus circa 1550 BC.[11] Algorithms were later used in ancient Hellenistic mathematics. Two examples are the Sieve of Eratosthenes, which was described in the *Introduction to Arithmetic* by Nicomachus,[82][12]:Ch 9.2 and the Euclidean algorithm, which was first described in *Euclid's Elements* (*c.* 300 BC).[12]:Ch 9.1

Discrete and distinguishable symbols

Tally-marks: To keep track of their flocks, their sacks of grain and their money the ancients used tallying: accumulating stones or marks scratched on sticks or making discrete symbols in clay. Through the Babylonian and Egyptian use of marks and symbols, eventually Roman numerals and the abacus evolved (Dilson, p. 16–41). Tally marks appear prominently in unary numeral system arithmetic used in Turing machine and Post–Turing machine computations.

Manipulation of symbols as "place holders" for numbers: algebra

Muhammad ibn Mūsā al-Khwārizmī, a Persian mathematician, wrote the *Al-jabr* in the 9th century. The terms "algorism" and "algorithm" are derived from the name al-Khwārizmī, while the term "algebra" is derived from the book *Al-jabr*. In Europe, the word "algorithm" was originally used to refer to the sets of rules and techniques used by Al-Khwarizmi to solve algebraic equations, before later being generalized to refer to any set of rules or techniques.[83] This eventually culminated in Leibniz's notion of the calculus ratiocinator (*c.* 1680):

> A good century and a half ahead of his time, Leibniz proposed an algebra of logic, an algebra that would specify the rules for manipulating logical concepts in the manner that ordinary algebra specifies the rules for manipulating numbers.[84]

Cryptographic algorithms

The first cryptographic algorithm for deciphering encrypted code was developed by Al-Kindi, a 9th-century Arab mathematician, in *A Manuscript On Deciphering Cryptographic Messages*. He gave the first description of cryptanalysis by frequency analysis, the earliest codebreaking algorithm.[13]

Mechanical contrivances with discrete states

The clock: Bolter credits the invention of the weight-driven clock as "The key invention [of Europe in the Middle Ages]", in particular, the verge escapement[85] that provides us with the tick and tock of a mechanical clock. "The accurate automatic machine"[86] led immediately to "mechanical automata" beginning in the 13th century and finally to "computational machines"—the difference engine and analytical engines of Charles Babbage and Countess Ada Lovelace, mid-19th century.[87] Lovelace is credited with the first creation of an algorithm intended for processing on a computer—Babbage's analytical engine, the first device considered a real Turing-complete computer instead of just a calculator—and is sometimes called "history's first programmer" as a result, though a full implementation of Babbage's second device would not be realized until decades after her lifetime.

Logical machines 1870 – Stanley Jevons' "logical abacus" and "logical machine": The technical problem was to reduce Boolean equations when presented in a form similar to what is now known as Karnaugh maps. Jevons (1880) describes first a simple "abacus" of "slips of wood furnished with pins, contrived so that any part or class of the [logical] combinations can be picked out mechanically ... More recently, however, I have reduced the system to a completely mechanical form, and have thus embodied the whole of the indirect process of inference in what may be called a *Logical Machine*" His machine came equipped with "certain moveable wooden rods" and "at the foot are 21 keys like those of a piano [etc.] ...". With this machine he could analyze a "syllogism or any other simple logical argument".[88]

This machine he displayed in 1870 before the Fellows of the Royal Society.[89] Another logician John Venn, however, in his 1881 *Symbolic Logic*, turned a jaundiced eye to this effort: "I have no high estimate myself of the interest or importance of what are sometimes called logical machines ... it does not seem to me that any contrivances at present known or likely to be discovered really deserve the name of logical machines"; see more at Algorithm characterizations. But not to be outdone he too presented "a plan somewhat analogous, I apprehend, to Prof. Jevon's *abacus* ... [And] [a]gain, corresponding to Prof. Jevons's logical machine, the following contrivance may be described. I prefer to call it merely a logical-diagram machine ... but I suppose that it could do very completely all that can be rationally expected of any logical machine".[90]

Jacquard loom, Hollerith punch cards, telegraphy and telephony – the electromechanical relay: Bell and Newell (1971) indicate that the Jacquard loom (1801), precursor to Hollerith cards (punch cards, 1887), and "telephone switching technologies" were the roots of a tree leading to the development of the first computers.[91] By the mid-19th century the telegraph, the precursor of the telephone, was in use throughout the world, its discrete and distinguishable encoding of letters as "dots and dashes" a common sound. By the late 19th century the ticker tape (*c.* 1870s) was in use, as was the use of Hollerith cards in the 1890 U.S. census. Then came the teleprinter (*c.* 1910) with its punched-paper use of Baudot code on tape.

Telephone-switching networks of electromechanical relays (invented 1835) was behind the work of George Stibitz (1937), the inventor of the digital adding device. As he worked in Bell Laboratories, he observed the "burdensome' use of mechanical calculators with gears. "He went home one evening in 1937 intending to test his idea... When the tinkering was over, Stibitz had constructed a binary adding device".[92]

The mathematician Martin Davis observes the particular importance of the electromechanical relay (with its two "binary states" *open* and *closed*):

> It was only with the development, beginning in the 1930s, of electromechanical calculators using electrical relays, that machines were built having the scope Babbage had envisioned."[93]

Mathematics during the 19th century up to the mid-20th century

Symbols and rules: In rapid succession, the mathematics of George Boole (1847, 1854), Gottlob Frege (1879), and Giuseppe Peano (1888–1889) reduced arithmetic to a sequence of symbols manipulated by rules. Peano's *The principles of arithmetic, presented by a new method* (1888) was "the first attempt at an axiomatization of mathematics in a symbolic language".[94]

But Heijenoort gives Frege (1879) this kudos: Frege's is "perhaps the most important single work ever written in logic. ... in which we see a " 'formula language', that is a *lingua characterica*, a language written with special symbols, "for pure thought", that is, free from rhetorical embellishments ... constructed from specific symbols that are manipulated according to definite rules".[95] The work of Frege was further simplified and amplified by Alfred North Whitehead and Bertrand Russell in their Principia Mathematica (1910–1913).

The paradoxes: At the same time a number of disturbing paradoxes appeared in the literature, in particular, the Burali-Forti paradox (1897), the Russell paradox (1902–03), and the Richard Paradox.[96] The resultant considerations led to Kurt Gödel's paper (1931)—he specifically cites the paradox of the liar—that completely reduces rules of recursion to numbers.

Effective calculability: In an effort to solve the Entscheidungsproblem defined precisely by Hilbert in 1928, mathematicians first set about to define what was meant by an "effective method" or "effective calculation" or "effective calculability" (i.e., a calculation that would succeed). In rapid succession the following appeared: Alonzo Church, Stephen Kleene and J.B. Rosser's λ-calculus[97] a finely honed definition of "general recursion" from the work of Gödel acting on suggestions of Jacques Herbrand (cf. Gödel's Princeton lectures of 1934) and subsequent simplifications by Kleene.[98] Church's proof[99] that the Entscheidungsproblem was unsolvable, Emil Post's definition of effective calculability as a worker mindlessly following a list of instructions to move left or right through a sequence of rooms and while there either mark or erase a paper or observe the paper and make a yes-no decision about the next instruction.[100] Alan Turing's proof of that the Entscheidungsproblem was unsolvable by use of his "a- [automatic-] machine"[101]—in effect almost identical to Post's "formulation", J. Barkley Rosser's definition of "effective method" in terms of "a machine".[102] Kleene's proposal of a precursor to "Church thesis" that he called "Thesis I",[103] and a few years later Kleene's renaming his Thesis "Church's Thesis"[104] and proposing "Turing's Thesis".[105]

Emil Post (1936) and Alan Turing (1936–37, 1939)

Emil Post (1936) described the actions of a "computer" (human being) as follows:

> "...two concepts are involved: that of a *symbol space* in which the work leading from problem to answer is to be carried out, and a fixed unalterable *set of directions*.

His symbol space would be

> "a two-way infinite sequence of spaces or boxes... The problem solver or worker is to move and work in this symbol space, being capable of being in, and operating in but one box at a time.... a box is to admit of but two possible conditions, i.e., being empty or unmarked, and having a single mark in it, say a vertical stroke.
>
> "One box is to be singled out and called the starting point. ...a specific problem is to be given in symbolic form by a finite number of boxes [i.e., INPUT] being marked with a stroke. Likewise, the answer [i.e., OUTPUT] is to be given in symbolic form by such a configuration of marked boxes...

"A set of directions applicable to a general problem sets up a deterministic process when applied to each specific problem. This process terminates only when it comes to the direction of type (C) [i.e., STOP]".[106] See more at Post–Turing machine

Alan Turing's work[107] preceded that of Stibitz (1937); it is unknown whether Stibitz knew of the work of Turing. Turing's biographer believed that Turing's use of a typewriter-like model derived from a youthful interest: "Alan had dreamt of inventing typewriters as a boy; Mrs. Turing had a typewriter, and he could well have begun by asking himself what was meant by calling a typewriter 'mechanical'".[108] Given the prevalence at the time of Morse code, telegraphy, ticker tape machines, and teletypewriters, it is quite possible that all were influences on Turing during his youth.

Alan Turing's statue at Bletchley Park

Turing—his model of computation is now called a Turing machine—begins, as did Post, with an analysis of a human computer that he whittles down to a simple set of basic motions and "states of mind". But he continues a step further and creates a machine as a model of computation of numbers.[109]

"Computing is normally done by writing certain symbols on paper. We may suppose this paper is divided into squares like a child's arithmetic book...I assume then that the computation is carried out on one-dimensional paper, i.e., on a tape divided into squares. I shall also suppose that the number of symbols which may be printed is finite...

"The behavior of the computer at any moment is determined by the symbols which he is observing, and his "state of mind" at that moment. We may suppose that there is a bound B to the number of symbols or squares that the computer can observe at one moment. If he wishes to observe more, he must use successive observations. We will also suppose that the number of states of mind which need be taken into account is finite...

"Let us imagine that the operations performed by the computer to be split up into 'simple operations' which are so elementary that it is not easy to imagine them further divided."[110]

Turing's reduction yields the following:

"The simple operations must therefore include:

"(a) Changes of the symbol on one of the observed squares
"(b) Changes of one of the squares observed to another square within L squares of one of the previously observed squares.

"It may be that some of these change necessarily invoke a change of state of mind. The most general single operation must, therefore, be taken to be one of the following:

"(A) A possible change (a) of symbol together with a possible change of state of mind.
"(B) A possible change (b) of observed squares, together with a possible change of state of mind"

"We may now construct a machine to do the work of this computer."[110]

A few years later, Turing expanded his analysis (thesis, definition) with this forceful expression of it:

"A function is said to be "effectively calculable" if its values can be found by some purely mechanical process. Though it is fairly easy to get an intuitive grasp of this idea, it is nevertheless desirable to have some more definite, mathematical expressible definition ... [he

discusses the history of the definition pretty much as presented above with respect to Gödel, Herbrand, Kleene, Church, Turing, and Post] ... We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and to an identification of computability † with effective calculability...

> "† We shall use the expression "computable function" to mean a function calculable by a machine, and we let "effectively calculable" refer to the intuitive idea without particular identification with any one of these definitions".[111]

J.B. Rosser (1939) and S.C. Kleene (1943)

J. Barkley Rosser defined an 'effective [mathematical] method' in the following manner (italicization added):

> "'Effective method' is used here in the rather special sense of a method each step of which is precisely determined and which is certain to produce the answer in a finite number of steps. With this special meaning, three different precise definitions have been given to date. [his footnote #5; see discussion immediately below]. The simplest of these to state (due to Post and Turing) says essentially that *an effective method of solving certain sets of problems exists if one can build a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer*. All three definitions are equivalent, so it doesn't matter which one is used. Moreover, the fact that all three are equivalent is a very strong argument for the correctness of any one." (Rosser 1939:225–226)

Rosser's footnote No. 5 references the work of (1) Church and Kleene and their definition of λ-definability, in particular, Church's use of it in his *An Unsolvable Problem of Elementary Number Theory* (1936); (2) Herbrand and Gödel and their use of recursion, in particular, Gödel's use in his famous paper *On Formally Undecidable Propositions of Principia Mathematica and Related Systems I* (1931); and (3) Post (1936) and Turing (1936–37) in their mechanism-models of computation.

Stephen C. Kleene defined as his now-famous "Thesis I" known as the Church–Turing thesis. But he did this in the following context (boldface in original):

> "12. *Algorithmic theories*... In setting up a complete algorithmic theory, what we do is to describe a procedure, performable for each set of values of the independent variables, which procedure necessarily terminates and in such manner that from the outcome we can read a definite answer, "yes" or "no," to the question, "is the predicate value true?"" (Kleene 1943:273)

History after 1950

A number of efforts have been directed toward further refinement of the definition of "algorithm", and activity is on-going because of issues surrounding, in particular, foundations of mathematics (especially the Church–Turing thesis) and philosophy of mind (especially arguments about artificial intelligence). For more, see Algorithm characterizations.

See also

- Abstract machine
- Algorithm engineering
- Algorithm characterizations

- Algorithmic bias
- Algorithmic composition
- Algorithmic entities
- Algorithmic synthesis
- Algorithmic technique
- Algorithmic topology
- Garbage in, garbage out
- *Introduction to Algorithms* (textbook)
- Government by algorithm
- List of algorithms
- List of algorithm general topics
- List of important publications in theoretical computer science – Algorithms
- Regulation of algorithms
- Theory of computation
  - Computability theory
  - Computational complexity theory
- Computational mathematics

Notes

1. "Definition of ALGORITHM" (https://www.merriam-webster.com/dictionary/algorithm). *Merriam-Webster Online Dictionary*. Archived (https://web.archive.org/web/20200214074446/https://www.merriam-webster.com/dictionary/algorithm) from the original on February 14, 2020. Retrieved November 14, 2019.
2. Blair, Ann, Duguid, Paul, Goeing, Anja-Silvia and Grafton, Anthony. Information: A Historical Companion, Princeton: Princeton University Press, 2021. p. 247
3. David A. Grossman, Ophir Frieder, *Information Retrieval: Algorithms and Heuristics*, 2nd edition, 2004, ISBN 1402030045
4. "Any classical mathematical algorithm, for example, can be described in a finite number of English words" (Rogers 1987:2).
5. Well defined with respect to the agent that executes the algorithm: "There is a computing agent, usually human, which can react to the instructions and carry out the computations" (Rogers 1987:2).
6. "an algorithm is a procedure for computing a *function* (with respect to some chosen notation for integers) ... this limitation (to numerical functions) results in no loss of generality", (Rogers 1987:1).
7. "An algorithm has zero or more inputs, i.e., quantities which are given to it initially before the algorithm begins" (Knuth 1973:5).
8. "A procedure which has all the characteristics of an algorithm except that it possibly lacks finiteness may be called a 'computational method'" (Knuth 1973:5).
9. "An algorithm has one or more outputs, i.e. quantities which have a specified relation to the inputs" (Knuth 1973:5).
10. Whether or not a process with random interior processes (not including the input) is an algorithm is debatable. Rogers opines that: "a computation is carried out in a discrete stepwise fashion, without the use of continuous methods or analogue devices ... carried forward deterministically, without resort to random methods or devices, e.g., dice" (Rogers 1987:2).
11. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*. Springer Science & Business Media. pp. 7–8. ISBN 9783642181924.
12. Cooke, Roger L. (2005). *The History of Mathematics: A Brief Course*. John Wiley & Sons. ISBN 978-1-118-46029-0.

13. Dooley, John F. (2013). *A Brief History of Cryptology and Cryptographic Algorithms*. Springer Science & Business Media. pp. 12–3. ISBN 9783319016283.
14. "Al-Khwarizmi biography" (http://www-history.mcs.st-andrews.ac.uk/Biographies/Al-Khwarizmi.html). *www-history.mcs.st-andrews.ac.uk*. Archived (https://web.archive.org/web/20190802091553/http://www-history.mcs.st-andrews.ac.uk/Biographies/Al-Khwarizmi.html) from the original on August 2, 2019. Retrieved May 3, 2017.
15. "Etymology of algorithm" (http://chambers.co.uk/search/?query=algorithm&title=21st). *Chambers Dictionary*. Archived (https://web.archive.org/web/20190331204600/https://chambers.co.uk/search/?query=algorithm&title=21st) from the original on March 31, 2019. Retrieved December 13, 2016.
16. Hogendijk, Jan P. (1998). "al-Khwarzimi" (https://web.archive.org/web/20090412193516/http://www.kennislink.nl/web/show?id=116543). *Pythagoras*. **38** (2): 4–5. Archived from the original (http://www.kennislink.nl/web/show?id=116543) on April 12, 2009.
17. Oaks, Jeffrey A. "Was al-Khwarizmi an applied algebraist?" (https://web.archive.org/web/20110718094835/http://facstaff.uindy.edu/~oaks/MHMC.htm). University of Indianapolis. Archived from the original (http://facstaff.uindy.edu/~oaks/MHMC.htm) on July 18, 2011. Retrieved May 30, 2008.
18. Brezina, Corona (2006). *Al-Khwarizmi: The Inventor Of Algebra* (https://books.google.com/books?id=955jPgAACAAJ). The Rosen Publishing Group. ISBN 978-1-4042-0513-0.
19. Foremost mathematical texts in history (http://www-history.mcs.st-and.ac.uk/Extras/Boyer_Foremost_Text.html) Archived (https://web.archive.org/web/20110609224820/http://www-history.mcs.st-and.ac.uk/Extras/Boyer_Foremost_Text.html) June 9, 2011, at the Wayback Machine, according to Carl B. Boyer.
20. "algorismic" (https://www.thefreedictionary.com/algorismic), *The Free Dictionary*, archived (https://web.archive.org/web/20191221200124/https://www.thefreedictionary.com/algorismic) from the original on December 21, 2019, retrieved November 14, 2019
21. *Oxford English Dictionary*, Third Edition, 2012 *s.v.* (http://www.oed.com/view/Entry/4959)
22. Sriram, M. S. (2005). "Algorithms in Indian Mathematics" (https://books.google.com/books?id=qfJdDwAAQBAJ&pg=PA153). In Emch, Gerard G.; Sridharan, R.; Srinivas, M. D. (eds.). *Contributions to the History of Indian Mathematics*. Springer. p. 153. ISBN 978-93-86279-25-5.
23. Mehri, Bahman (2017). "From Al-Khwarizmi to Algorithm". *Olympiads in Informatics*. **11** (2): 71–74. doi:10.15388/ioi.2017.special.11 (https://doi.org/10.15388%2Fioi.2017.special.11).
24. "Abu Jafar Muhammad ibn Musa al-Khwarizmi" (http://members.peak.org/~jeremy/calculators/alKwarizmi.html). *members.peak.org*. Archived (https://web.archive.org/web/20190821232118/http://members.peak.org/~jeremy/calculators/alKwarizmi.html) from the original on August 21, 2019. Retrieved November 14, 2019.
25. Kleene 1943 in Davis 1965:274
26. Rosser 1939 in Davis 1965:225
27. Stone 1973:4
28. Simanowski, Roberto (2018). *The Death Algorithm and Other Digital Dilemmas* (https://books.google.com/books?id=RJV5DwAAQBAJ). Untimely Meditations. Vol. 14. Translated by Chase, Jefferson. Cambridge, Massachusetts: MIT Press. p. 147. ISBN 9780262536370. Archived (https://web.archive.org/web/20191222120705/https://books.google.com/books?id=RJV5DwAAQBAJ) from the original on December 22, 2019. Retrieved May 27, 2019. "[...] the next level of abstraction of central bureaucracy: globally operating algorithms."
29. Dietrich, Eric (1999). "Algorithm". In Wilson, Robert Andrew; Keil, Frank C. (eds.). *The MIT Encyclopedia of the Cognitive Sciences* (https://books.google.com/books?id=-wt1aZrGXLYC). MIT Cognet library. Cambridge, Massachusetts: MIT Press (published 2001). p. 11. ISBN 9780262731447. Retrieved July 22, 2020. "An algorithm is a recipe, method, or technique for doing something."
30. Stone simply requires that "it must terminate in a finite number of steps" (Stone 1973:7–8).
31. Boolos and Jeffrey 1974,1999:19

32. cf Stone 1972:5
33. Knuth 1973:7 states: "In practice we not only want algorithms, we want *good* algorithms ... one criterion of goodness is the length of time taken to perform the algorithm ... other criteria are the adaptability of the algorithm to computers, its simplicity, and elegance, etc."
34. cf Stone 1973:6
35. Stone 1973:7–8 states that there must be, "...a procedure that a robot [i.e., computer] can follow in order to determine precisely how to obey the instruction". Stone adds finiteness of the process, and definiteness (having no ambiguity in the instructions) to this definition.
36. Knuth, loc. cit
37. Minsky 1967, p. 105
38. Gurevich 2000:1, 3
39. Sipser 2006:157
40. Goodrich, Michael T.; Tamassia, Roberto (2002), *Algorithm Design: Foundations, Analysis, and Internet Examples* (http://ww3.algorithmdesign.net/ch00-front.html), John Wiley & Sons, Inc., ISBN 978-0-471-38365-9, archived (https://web.archive.org/web/20150428201622/http://ww3.algorithmdesign.net/ch00-front.html) from the original on April 28, 2015, retrieved June 14, 2018
41. Knuth 1973:7
42. Chaitin 2005:32
43. Rogers 1987:1–2
44. In his essay "Calculations by Man and Machine: Conceptual Analysis" Seig 2002:390 credits this distinction to Robin Gandy, cf Wilfred Seig, et al., 2002 *Reflections on the foundations of mathematics: Essays in honor of Solomon Feferman*, Association for Symbolic Logic, A.K. Peters Ltd, Natick, MA.
45. cf Gandy 1980:126, Robin Gandy *Church's Thesis and Principles for Mechanisms* appearing on pp. 123–148 in J. Barwise et al. 1980 *The Kleene Symposium*, North-Holland Publishing Company.
46. A "robot": "A computer is a robot that performs any task that can be described as a sequence of instructions." cf Stone 1972:3
47. Lambek's "abacus" is a "countably infinite number of locations (holes, wires etc.) together with an unlimited supply of counters (pebbles, beads, etc.). The locations are distinguishable, the counters are not". The holes have unlimited capacity, and standing by is an agent who understands and is able to carry out the list of instructions" (Lambek 1961:295). Lambek references Melzak who defines his Q-machine as "an indefinitely large number of locations ... an indefinitely large supply of counters distributed among these locations, a program, and an operator whose sole purpose is to carry out the program" (Melzak 1961:283). B-B-J (loc. cit.) add the stipulation that the holes are "capable of holding any number of stones" (p. 46). Both Melzak and Lambek appear in *The Canadian Mathematical Bulletin*, vol. 4, no. 3, September 1961.
48. If no confusion results, the word "counters" can be dropped, and a location can be said to contain a single "number".
49. "We say that an instruction is *effective* if there is a procedure that the robot can follow in order to determine precisely how to obey the instruction." (Stone 1972:6)
50. cf Minsky 1967: Chapter 11 "Computer models" and Chapter 14 "Very Simple Bases for Computability" pp. 255–281, in particular,
51. cf Knuth 1973:3.
52. But always preceded by IF-THEN to avoid improper subtraction.
53. Knuth 1973:4
54. Stone 1972:5. Methods for extracting roots are not trivial: see Methods of computing square roots.

55. Leeuwen, Jan (1990). *Handbook of Theoretical Computer Science: Algorithms and complexity. Volume A* (https://books.google.com/books?id=-X39_rA3VSQC). Elsevier. p. 85. ISBN 978-0-444-88071-0.
56. John G. Kemeny and Thomas E. Kurtz 1985 *Back to Basic: The History, Corruption, and Future of the Language*, Addison-Wesley Publishing Company, Inc. Reading, MA, ISBN 0-201-13433-0.
57. Tausworthe 1977:101
58. Tausworthe 1977:142
59. Knuth 1973 section 1.2.1, expanded by Tausworthe 1977 at pages 100ff and Chapter 9.1
60. cf Tausworthe 1977
61. Heath 1908:300; Hawking's Dover 2005 edition derives from Heath.
62. " 'Let CD, measuring BF, leave FA less than itself.' This is a neat abbreviation for saying, measure along BA successive lengths equal to CD until a point F is reached such that the length FA remaining is less than CD; in other words, let BF be the largest exact multiple of CD contained in BA" (Heath 1908:297)
63. For modern treatments using division in the algorithm, see Hardy and Wright 1979:180, Knuth 1973:2 (Volume 1), plus more discussion of Euclid's algorithm in Knuth 1969:293–297 (Volume 2).
64. Euclid covers this question in his Proposition 1.
65. "Euclid's Elements, Book VII, Proposition 2" (http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html). Aleph0.clarku.edu. Archived (https://web.archive.org/web/20120524074919/http://aleph0.clarku.edu/~djoyce/java/elements/bookVII/propVII2.html) from the original on May 24, 2012. Retrieved May 20, 2012.
66. While this notion is in widespread use, it cannot be defined precisely.
67. Knuth 1973:13–18. He credits "the formulation of algorithm-proving in terms of assertions and induction" to R W. Floyd, Peter Naur, C.A.R. Hoare, H.H. Goldstine and J. von Neumann. Tausworth 1977 borrows Knuth's Euclid example and extends Knuth's method in section 9.1 *Formal Proofs* (pp. 288–298).
68. Tausworthe 1997:294
69. cf Knuth 1973:7 (Vol. I), and his more-detailed analyses on pp. 1969:294–313 (Vol II).
70. Breakdown occurs when an algorithm tries to compact itself. Success would solve the Halting problem.
71. Kriegel, Hans-Peter; Schubert, Erich; Zimek, Arthur (2016). "The (black) art of run-time evaluation: Are we comparing algorithms or implementations?". *Knowledge and Information Systems*. **52** (2): 341–378. doi:10.1007/s10115-016-1004-2 (https://doi.org/10.1007%2Fs10115-016-1004-2). ISSN 0219-1377 (https://www.worldcat.org/issn/0219-1377). S2CID 40772241 (https://api.semanticscholar.org/CorpusID:40772241).
72. Gillian Conahan (January 2013). "Better Math Makes Faster Data Networks" (http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks). discovermagazine.com. Archived (https://web.archive.org/web/20140513212427/http://discovermagazine.com/2013/jan-feb/34-better-math-makes-faster-data-networks) from the original on May 13, 2014. Retrieved May 13, 2014.
73. Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price, "ACM-SIAM Symposium On Discrete Algorithms (SODA) (http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) Archived (https://web.archive.org/web/20130704180806/http://siam.omnibooksonline.com/2012SODA/data/papers/500.pdf) July 4, 2013, at the Wayback Machine, Kyoto, January 2012. See also the sFFT Web Page (http://groups.csail.mit.edu/netmit/sFFT/) Archived (https://web.archive.org/web/20120221145740/http://groups.csail.mit.edu/netmit/sFFT/) February 21, 2012, at the Wayback Machine.
74. Kowalski 1979

75. Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). *Knapsack Problems | Hans Kellerer | Springer* (https://www.springer.com/us/book/9783540402862). Springer. doi:10.1007/978-3-540-24777-7 (https://doi.org/10.1007%2F978-3-540-24777-7). ISBN 978-3-540-40286-2. S2CID 28836720 (https://api.semanticscholar.org/CorpusID:28836720). Archived (https://web.archive.org/web/20171018181055/https://www.springer.com/us/book/9783540402862) from the original on October 18, 2017. Retrieved September 19, 2017.
76. Carroll, Sue; Daughtrey, Taz (July 4, 2007). *Fundamental Concepts for the Software Quality Engineer* (https://books.google.com/books?id=bz_cl3B05IcC&pg=PA282). American Society for Quality. pp. 282 et seq. ISBN 978-0-87389-720-4.
77. For instance, the volume of a convex polytope (described using a membership oracle) can be approximated to high accuracy by a randomized polynomial time algorithm, but not by a deterministic one: see Dyer, Martin; Frieze, Alan; Kannan, Ravi (January 1991), "A Random Polynomial-time Algorithm for Approximating the Volume of Convex Bodies", *J. ACM*, **38** (1): 1–17, CiteSeerX 10.1.1.145.4600 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.145.4600), doi:10.1145/102782.102783 (https://doi.org/10.1145%2F102782.102783), S2CID 13268711 (https://api.semanticscholar.org/CorpusID:13268711).
78. George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.
79. Tsypkin (1971). *Adaptation and learning in automatic systems* (https://books.google.com/books?id=sgDHJlafMskC&pg=PA54). Academic Press. p. 54. ISBN 978-0-08-095582-7.
80. Knuth, Donald E. (1972). "Ancient Babylonian Algorithms" (https://web.archive.org/web/20121224100137/http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF). *Commun. ACM*. **15** (7): 671–677. doi:10.1145/361454.361514 (https://doi.org/10.1145%2F361454.361514). ISSN 0001-0782 (https://www.worldcat.org/issn/0001-0782). S2CID 7829945 (https://api.semanticscholar.org/CorpusID:7829945). Archived from the original (http://steiner.math.nthu.edu.tw/disk5/js/computer/1.pdf) (PDF) on December 24, 2012.
81. Aaboe, Asger (2001), *Episodes from the Early History of Astronomy*, New York: Springer, pp. 40–62, ISBN 978-0-387-95136-2
82. Ast, Courtney. "Eratosthenes" (http://www.math.wichita.edu/history/men/eratosthenes.html). Wichita State University: Department of Mathematics and Statistics. Archived (https://web.archive.org/web/20150227150653/http://www.math.wichita.edu/history/men/eratosthenes.html) from the original on February 27, 2015. Retrieved February 27, 2015.
83. Chabert, Jean-Luc (2012). *A History of Algorithms: From the Pebble to the Microchip*. Springer Science & Business Media. p. 2. ISBN 9783642181924.
84. Davis 2000:18
85. Bolter 1984:24
86. Bolter 1984:26
87. Bolter 1984:33–34, 204–206.
88. All quotes from W. Stanley Jevons 1880 *Elementary Lessons in Logic: Deductive and Inductive*, Macmillan and Co., London and New York. Republished as a googlebook; cf Jevons 1880:199–201. Louis Couturat 1914 *the Algebra of Logic*, The Open Court Publishing Company, Chicago and London. Republished as a googlebook; cf Couturat 1914:75–76 gives a few more details; he compares this to a typewriter as well as a piano. Jevons states that the account is to be found at January 20, 1870 *The Proceedings of the Royal Society*.
89. Jevons 1880:199–200
90. All quotes from John Venn 1881 *Symbolic Logic*, Macmillan and Co., London. Republished as a googlebook. cf Venn 1881:120–125. The interested reader can find a deeper explanation in those pages.
91. Bell and Newell diagram 1971:39, cf. Davis 2000
92. * Melina Hill, Valley News Correspondent, *A Tinkerer Gets a Place in History*, Valley News West Lebanon NH, Thursday, March 31, 1983, p. 13.
93. Davis 2000:14

94. van Heijenoort 1967:81ff
95. van Heijenoort's commentary on Frege's *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought* in van Heijenoort 1967:1
96. Dixon 1906, cf. Kleene 1952:36–40
97. cf. footnote in Alonzo Church 1936a in Davis 1965:90 and 1936b in Davis 1965:110
98. Kleene 1935–6 in Davis 1965:237ff, Kleene 1943 in Davis 1965:255ff
99. Church 1936 in Davis 1965:88ff
100. cf. "Finite Combinatory Processes – formulation 1", Post 1936 in Davis 1965:289–290
101. Turing 1936–37 in Davis 1965:116ff
102. Rosser 1939 in Davis 1965:226
103. Kleene 1943 in Davis 1965:273–274
104. Kleene 1952:300, 317
105. Kleene 1952:376
106. Turing 1936–37 in Davis 1965:289–290
107. Turing 1936 in Davis 1965, Turing 1939 in Davis 1965:160
108. Hodges, p. 96
109. Turing 1936–37:116
110. Turing 1936–37 in Davis 1965:136
111. Turing 1939 in Davis 1965:160

Bibliography

- Axt, P (1959). "On a Subrecursive Hierarchy and Primitive Recursive Degrees" (https://doi.org/10.2307%2F1993169). *Transactions of the American Mathematical Society*. **92** (1): 85–105. doi:10.2307/1993169 (https://doi.org/10.2307%2F1993169). JSTOR 1993169 (https://www.jstor.org/stable/1993169).
- Bell, C. Gordon and Newell, Allen (1971), *Computer Structures: Readings and Examples*, McGraw–Hill Book Company, New York. ISBN 0-07-004357-4.
- Blass, Andreas; Gurevich, Yuri (2003). "Algorithms: A Quest for Absolute Definitions" (http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF). *Bulletin of European Association for Theoretical Computer Science*. **81**. Includes an excellent bibliography of 56 references.
- Bolter, David J. (1984). *Turing's Man: Western Culture in the Computer Age* (1984 ed.). Chapel Hill, NC: The University of North Carolina Press. ISBN 978-0-8078-1564-9., ISBN 0-8078-4108-0
- Boolos, George; Jeffrey, Richard (1999) [1974]. *Computability and Logic* (https://archive.org/details/computabilitylog0000bool_r8y9) (4th ed.). Cambridge University Press, London. ISBN 978-0-521-20402-6.: cf. Chapter 3 *Turing machines* where they discuss "certain enumerable sets not effectively (mechanically) enumerable".
- Burgin, Mark (2004). *Super-Recursive Algorithms*. Springer. ISBN 978-0-387-95569-8.
- Campagnolo, M.L., Moore, C., and Costa, J.F. (2000) An analog characterization of the subrecursive functions. In *Proc. of the 4th Conference on Real Numbers and Computers*, Odense University, pp. 91–109
- Church, Alonzo (1936). "An Unsolvable Problem of Elementary Number Theory". *The American Journal of Mathematics*. **58** (2): 345–363. doi:10.2307/2371045 (https://doi.org/10.2307%2F2371045). JSTOR 2371045 (https://www.jstor.org/stable/2371045). Reprinted in *The Undecidable*, p. 89ff. The first expression of "Church's Thesis". See in particular page 100 (*The Undecidable*) where he defines the notion of "effective calculability" in terms of "an algorithm", and he uses the word "terminates", etc.

- Church, Alonzo (1936). "A Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (1): 40–41. doi:10.2307/2269326 (https://doi.org/10.2307%2F2269326). JSTOR 2269326 (https://www.jstor.org/stable/2269326). S2CID 42323521 (https://api.semanticscholar.org/CorpusID:42323521). Church, Alonzo (1936). "Correction to a Note on the Entscheidungsproblem". *The Journal of Symbolic Logic*. **1** (3): 101–102. doi:10.2307/2269030 (https://doi.org/10.2307%2F2269030). JSTOR 2269030 (https://www.jstor.org/stable/2269030). S2CID 5557237 (https://api.semanticscholar.org/CorpusID:5557237). Reprinted in *The Undecidable*, p. 110ff. Church shows that the Entscheidungsproblem is unsolvable in about 3 pages of text and 3 pages of footnotes.
- Daffa', Ali Abdullah al- (1977). *The Muslim contribution to mathematics*. London: Croom Helm. ISBN 978-0-85664-464-1.
- Davis, Martin (1965). *The Undecidable: Basic Papers On Undecidable Propositions, Unsolvable Problems and Computable Functions* (https://archive.org/details/undecidablebasic0000davi). New York: Raven Press. ISBN 978-0-486-43228-1. Davis gives commentary before each article. Papers of Gödel, Alonzo Church, Turing, Rosser, Kleene, and Emil Post are included; those cited in the article are listed here by author's name.
- Davis, Martin (2000). *Engines of Logic: Mathematicians and the Origin of the Computer*. New York: W.W. Nortion. ISBN 978-0-393-32229-3. Davis offers concise biographies of Leibniz, Boole, Frege, Cantor, Hilbert, Gödel and Turing with von Neumann as the show-stealing villain. Very brief bios of Joseph-Marie Jacquard, Babbage, Ada Lovelace, Claude Shannon, Howard Aiken, etc.
- This article incorporates public domain material from the NIST document: Black, Paul E. "algorithm" (https://xlinux.nist.gov/dads/HTML/algorithm.html). *Dictionary of Algorithms and Data Structures*.
- Dean, Tim (2012). "Evolution and moral diversity" (https://doi.org/10.4148%2Fbiyclc.v7i0.1775). *Baltic International Yearbook of Cognition, Logic and Communication*. **7**. doi:10.4148/biyclc.v7i0.1775 (https://doi.org/10.4148%2Fbiyclc.v7i0.1775).
- Dennett, Daniel (1995). *Darwin's Dangerous Idea* (https://archive.org/details/darwinsdangerous0000denn). *Complexity*. Vol. 2. New York: Touchstone/Simon & Schuster. pp. 32 (https://archive.org/details/darwinsdangerous0000denn/page/32)–36. Bibcode:1996Cmplx...2a..32M (https://ui.adsabs.harvard.edu/abs/1996Cmplx...2a..32M). doi:10.1002/(SICI)1099-0526(199609/10)2:1<32::AID-CPLX8>3.0.CO;2-H (https://doi.org/10.1002%2F%28SICI%2910 99-0526%28199609%2F10%292%3A1%3C32%3A%3AAID-CPLX8%3E3.0.CO%3B2-H). ISBN 978-0-684-80290-9.
- Dilson, Jesse (2007). *The Abacus* (https://archive.org/details/abacusworldsfirs0000dils) ((1968, 1994) ed.). St. Martin's Press, NY. ISBN 978-0-312-10409-2., ISBN 0-312-10409-X
- Yuri Gurevich, *Sequential Abstract State Machines Capture Sequential Algorithms* (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.146.3017&rep=rep1&type=pdf), ACM Transactions on Computational Logic, Vol 1, no 1 (July 2000), pp. 77–111. Includes bibliography of 33 sources.
- van Heijenoort, Jean (2001). *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931* ((1967) ed.). Harvard University Press, Cambridge. ISBN 978-0-674-32449-7., 3rd edition 1976[?], ISBN 0-674-32449-8 (pbk.)
- Hodges, Andrew (1983). *Alan Turing: The Enigma*. *Physics Today*. Vol. 37. New York: Simon and Schuster. pp. 107–108. Bibcode:1984PhT....37k.107H (https://ui.adsabs.harvard.edu/abs/1984PhT....37k.107H). doi:10.1063/1.2915935 (https://doi.org/10.1063%2F1.2915935). ISBN 978-0-671-49207-6., ISBN 0-671-49207-1. Cf. Chapter "The Spirit of Truth" for a history leading to, and a discussion of, his proof.

- Kleene, Stephen C. (1936). "General Recursive Functions of Natural Numbers" (https://web.archive.org/web/20140903092121/http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1). *Mathematische Annalen*. **112** (5): 727–742. doi:10.1007/BF01565439 (https://doi.org/10.1007%2FBF01565439). S2CID 120517999 (https://api.semanticscholar.org/CorpusID:120517999). Archived from the original (http://gdz.sub.uni-goettingen.de/index.php?id=11&PPN=GDZPPN002278499&L=1) on September 3, 2014. Retrieved September 30, 2013. Presented to the American Mathematical Society, September 1935. Reprinted in *The Undecidable*, p. 237ff. Kleene's definition of "general recursion" (known now as mu-recursion) was used by Church in his 1935 paper *An Unsolvable Problem of Elementary Number Theory* that proved the "decision problem" to be "undecidable" (i.e., a negative result).
- Kleene, Stephen C. (1943). "Recursive Predicates and Quantifiers" (https://doi.org/10.2307%2F1990131). *Transactions of the American Mathematical Society*. **53** (1): 41–73. doi:10.2307/1990131 (https://doi.org/10.2307%2F1990131). JSTOR 1990131 (https://www.jstor.org/stable/1990131). Reprinted in *The Undecidable*, p. 255ff. Kleene refined his definition of "general recursion" and proceeded in his chapter "12. Algorithmic theories" to posit "Thesis I" (p. 274); he would later repeat this thesis (in Kleene 1952:300) and name it "Church's Thesis" (Kleene 1952:317) (i.e., the Church thesis).
- Kleene, Stephen C. (1991) [1952]. *Introduction to Metamathematics* (Tenth ed.). North-Holland Publishing Company. ISBN 978-0-7204-2103-3.
- Knuth, Donald (1997). *Fundamental Algorithms, Third Edition*. Reading, Massachusetts: Addison–Wesley. ISBN 978-0-201-89683-1.
- Knuth, Donald (1969). *Volume 2/Seminumerical Algorithms, The Art of Computer Programming First Edition*. Reading, Massachusetts: Addison–Wesley.
- Kosovsky, N.K. *Elements of Mathematical Logic and its Application to the theory of Subrecursive Algorithms*, LSU Publ., Leningrad, 1981
- Kowalski, Robert (1979). "Algorithm=Logic+Control". *Communications of the ACM*. **22** (7): 424–436. doi:10.1145/359131.359136 (https://doi.org/10.1145%2F359131.359136). S2CID 2509896 (https://api.semanticscholar.org/CorpusID:2509896).
- A.A. Markov (1954) *Theory of algorithms*. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954 [i.e., Jerusalem, Israel Program for Scientific Translations, 1961; available from the Office of Technical Services, U.S. Dept. of Commerce, Washington] Description 444 p. 28 cm. Added t.p. in Russian Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algerifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.]
- Minsky, Marvin (1967). *Computation: Finite and Infinite Machines* (https://archive.org/details/computationfinit0000mins) (First ed.). Prentice-Hall, Englewood Cliffs, NJ. ISBN 978-0-13-165449-5. Minsky expands his "...idea of an algorithm – an effective procedure..." in chapter 5.1 *Computability, Effective Procedures and Algorithms. Infinite machines.*
- Post, Emil (1936). "Finite Combinatory Processes, Formulation I". *The Journal of Symbolic Logic*. **1** (3): 103–105. doi:10.2307/2269031 (https://doi.org/10.2307%2F2269031). JSTOR 2269031 (https://www.jstor.org/stable/2269031). S2CID 40284503 (https://api.semanticscholar.org/CorpusID:40284503). Reprinted in *The Undecidable*, pp. 289ff. Post defines a simple algorithmic-like process of a man writing marks or erasing marks and going from box to box and eventually halting, as he follows a list of simple instructions. This is cited by Kleene as one source of his "Thesis I", the so-called Church–Turing thesis.
- Rogers, Jr, Hartley (1987). *Theory of Recursive Functions and Effective Computability*. The MIT Press. ISBN 978-0-262-68052-3.

- Rosser, J.B. (1939). "An Informal Exposition of Proofs of Godel's Theorem and Church's Theorem". *Journal of Symbolic Logic*. **4** (2): 53–60. doi:10.2307/2269059 (https://doi.org/10.2307%2F2269059). JSTOR 2269059 (https://www.jstor.org/stable/2269059). S2CID 39499392 (https://api.semanticscholar.org/CorpusID:39499392). Reprinted in *The Undecidable*, p. 223ff. Herein is Rosser's famous definition of "effective method": "...a method each step of which is precisely predetermined and which is certain to produce the answer in a finite number of steps... a machine which will then solve any problem of the set with no human intervention beyond inserting the question and (later) reading the answer" (p. 225–226, *The Undecidable*)
- Santos-Lang, Christopher (2014). "Moral Ecology Approaches to Machine Ethics" (http://grinfree.com/MoralEcology.pdf) (PDF). In van Rysewyk, Simon; Pontier, Matthijs (eds.). *Machine Medical Ethics*. Intelligent Systems, Control and Automation: Science and Engineering. Vol. 74. Switzerland: Springer. pp. 111–127. doi:10.1007/978-3-319-08108-3_8 (https://doi.org/10.1007%2F978-3-319-08108-3_8). ISBN 978-3-319-08107-6.
- Scott, Michael L. (2009). *Programming Language Pragmatics* (3rd ed.). Morgan Kaufmann Publishers/Elsevier. ISBN 978-0-12-374514-9.
- Sipser, Michael (2006). *Introduction to the Theory of Computation* (https://archive.org/details/introductiontoth00sips). PWS Publishing Company. ISBN 978-0-534-94728-6.
- Sober, Elliott; Wilson, David Sloan (1998). *Unto Others: The Evolution and Psychology of Unselfish Behavior* (https://archive.org/details/untoothersevolut00sobe). Cambridge: Harvard University Press. ISBN 9780674930469.
- Stone, Harold S. (1972). *Introduction to Computer Organization and Data Structures* (1972 ed.). McGraw-Hill, New York. ISBN 978-0-07-061726-1. Cf. in particular the first chapter titled: *Algorithms, Turing Machines, and Programs*. His succinct informal definition: "...any sequence of instructions that can be obeyed by a robot, is called an *algorithm*" (p. 4).
- Tausworthe, Robert C (1977). *Standardized Development of Computer Software Part 1 Methods*. Englewood Cliffs NJ: Prentice–Hall, Inc. ISBN 978-0-13-842195-3.
- Turing, Alan M. (1936–37). "On Computable Numbers, With An Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*. Series 2. **42**: 230–265. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712).. Corrections, ibid, vol. 43(1937) pp. 544–546. Reprinted in *The Undecidable*, p. 116ff. Turing's famous paper completed as a Master's dissertation while at King's College Cambridge UK.
- Turing, Alan M. (1939). "Systems of Logic Based on Ordinals". *Proceedings of the London Mathematical Society*. **45**: 161–228. doi:10.1112/plms/s2-45.1.161 (https://doi.org/10.1112%2Fplms%2Fs2-45.1.161). hdl:21.11116/0000-0001-91CE-3 (https://hdl.handle.net/21.11116%2F0000-0001-91CE-3). Reprinted in *The Undecidable*, pp. 155ff. Turing's paper that defined "the oracle" was his PhD thesis while at Princeton.
- United States Patent and Trademark Office (2006), *2106.02 **>Mathematical Algorithms: 2100 Patentability* (http://www.uspto.gov/web/offices/pac/mpep/documents/2100_2106_02.htm), Manual of Patent Examining Procedure (MPEP). Latest revision August 2006

Further reading

- Bellah, Robert Neelly (1985). *Habits of the Heart: Individualism and Commitment in American Life* (https://books.google.com/books?id=XsUojihVZQcC). Berkeley: University of California Press. ISBN 978-0-520-25419-0.
- Berlinski, David (2001). *The Advent of the Algorithm: The 300-Year Journey from an Idea to the Computer* (https://archive.org/details/adventofalgorith0000berl). Harvest Books. ISBN 978-0-15-601391-8.
- Chabert, Jean-Luc (1999). *A History of Algorithms: From the Pebble to the Microchip*. Springer Verlag. ISBN 978-3-540-63369-3.

- Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein (2009). *Introduction To Algorithms* (3rd ed.). MIT Press. ISBN 978-0-262-03384-8.
- Harel, David; Feldman, Yishai (2004). *Algorithmics: The Spirit of Computing*. Addison-Wesley. ISBN 978-0-321-11784-7.
- Hertzke, Allen D.; McRorie, Chris (1998). "The Concept of Moral Ecology". In Lawler, Peter Augustine; McConkey, Dale (eds.). *Community and Political Thought Today*. Westport, CT: Praeger.
- Knuth, Donald E. (2000). *Selected Papers on Analysis of Algorithms (http://www-cs-faculty.stanford.edu/~uno/aa.html)*. Stanford, California: Center for the Study of Language and Information.
- Knuth, Donald E. (2010). *Selected Papers on Design of Algorithms (http://www-cs-faculty.stanford.edu/~uno/da.html)*. Stanford, California: Center for the Study of Language and Information.
- Wallach, Wendell; Allen, Colin (November 2008). *Moral Machines: Teaching Robots Right from Wrong*. US: Oxford University Press. ISBN 978-0-19-537404-9.
- Bleakley, Chris (2020). *Poems that Solve Puzzles: The History and Science of Algorithms* (https://books.google.com/books?id=3pr5DwAAQBAJ). Oxford University Press. ISBN 978-0-19-885373-2.

External links

- "Algorithm" (https://www.encyclopediaofmath.org/index.php?title=Algorithm), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]
- Algorithms (https://curlie.org/Computers/Algorithms/) at Curlie
- Weisstein, Eric W. "Algorithm" (https://mathworld.wolfram.com/Algorithm.html). *MathWorld*.
- Dictionary of Algorithms and Data Structures (https://www.nist.gov/dads/) – National Institute of Standards and Technology

Algorithm repositories

- The Stony Brook Algorithm Repository (http://www.cs.sunysb.edu/~algorith/) – State University of New York at Stony Brook
- Collected Algorithms of the ACM (http://calgo.acm.org/) – Association for Computing Machinery
- The Stanford GraphBase (http://www-cs-staff.stanford.edu/~knuth/sgb.html) – Stanford University

---

Smart contract

A **smart contract** is a computer program or a transaction protocol that is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement.[1][2][3][4] The objectives of smart contracts are the reduction of need for trusted intermediators, arbitrations costs, fraud losses, as well as the reduction of malicious and accidental exceptions.[5][2] Smart contracts are commonly associated with cryptocurrencies, and the smart contracts introduced by Ethereum are generally considered a fundamental building block for decentralized finance (DeFi) applications.[6][7]

Vending machines are mentioned as the oldest piece of technology equivalent to smart contract implementation.[3] The original Ethereum white paper by Vitalik Buterin in 2014[8] describes the Bitcoin protocol as a weak version of the smart contract concept as originally defined by Nick Szabo, and proposes a stronger version based on the Solidity language, which is Turing complete. Since Bitcoin, various cryptocurrencies support scripting languages which allow for more advanced smart contracts between untrusted parties.[9] Smart contracts should be distinguished from smart legal contracts, where the latter refers to traditional natural-language legally-binding agreements that have selected terms expressed and implemented in machine-readable code.[10][11][12]

Contents

Etymology

Smart contracts were first proposed in the early 1990s by Nick Szabo, who coined the term, using it to refer to "a set of promises, specified in digital form, including protocols within which the parties perform on these promises".[13][14] In 1998, the term was used to describe objects in rights management service layer of the system *The Stanford Infobus*, which was a part of Stanford Digital Library Project.[1]

Legal status of smart contracts

A smart contract does not necessarily constitute a valid binding agreement at law.[15] Some legal academics claim that smart contracts are not legal agreements, but rather means of performing obligations deriving from other agreements[16] such as technological means for the automation of

payment obligations[17] or obligations consisting in the transfer of tokens or cryptocurrencies. Additionally, other scholars have argued that the imperative or declarative nature of programming languages can impact the legal validity of smart contracts.[18]

Since the 2015 launch of the Ethereum blockchain,[19] the term "smart contract" has been more specifically applied toward the notion of general purpose computation that takes place on a blockchain or distributed ledger. The US National Institute of Standards and Technology describes a "smart contract" as a "collection of code and data (sometimes referred to as functions and state) that is deployed using cryptographically signed transactions on the blockchain network".[20] In this interpretation, used for example by the Ethereum Foundation[8] or IBM,[21] a smart contract is not necessarily related to the classical concept of a contract, but can be any kind of computer program. A smart contract also can be regarded as a secured stored procedure as its execution and codified effects like the transfer of some value between parties are strictly enforced and can not be manipulated, after a transaction with specific contract details is stored into a blockchain or distributed ledger. That's because the actual execution of contracts is controlled and audited by the platform, not by any arbitrary server-side programs connecting to the platform.[22][23]

In 2017, by implementing the Decree on Development of Digital Economy, Belarus has become the first-ever country to legalize smart contracts. Belarusian lawyer Denis Aleinikov is considered to be the author of a smart contract legal concept introduced by the decree.[24]

In 2018, a US Senate report said: "While smart contracts might sound new, the concept is rooted in basic contract law. Usually, the judicial system adjudicates contractual disputes and enforces terms, but it is also common to have another arbitration method, especially for international transactions. With smart contracts, a program enforces the contract built into the code."[25] A number of states in the US have passed legislation on the use of smart contracts, such as Arizona,[26] Nevada,[27] Tennessee,[28] and Wyoming.[29] And in April 2020, Iowa's House of Representatives passed a bill legally recognizing smart contracts in the state.[30]

In April 2021, the UK Jurisdiction Taskforce (UKJT) published the Digital Dispute Resolution Rules (the Digital DR Rules) to help enable the rapid resolution of blockchain and crypto legal disputes in Britain.[31]

Workings

Similar to a transfer of value on a blockchain, deployment of a smart contract on a blockchain occurs by sending a transaction from a wallet for the blockchain.[32] The transaction includes the compiled code for the smart contract as well as a special receiver address.[32] That transaction must then be included in a block that is added to the blockchain, at which point the smart contract's code will execute to establish the initial state of the smart contract.[32] Byzantine fault-tolerant algorithms secure the smart contract in a decentralized way from attempts to tamper with it. Once a smart contract is deployed, it cannot be updated.[33] Smart contracts on a blockchain can store arbitrary state and execute arbitrary computations. End clients interact with a smart contract through transactions. Such transactions with a smart contract can invoke other smart contracts. These transactions might result in changing the state and sending coins from one smart contract to another or from one account to another.[33]

The most popular blockchain for running smart contracts is Ethereum.[34] On Ethereum, smart contracts are typically written in a Turing-complete programming language called Solidity,[35] and compiled into low-level bytecode to be executed by the Ethereum Virtual Machine.[36] Due to the halting problem and other security problems, Turing-completeness is considered to be a risk and is deliberately avoided by languages like Vyper.[37][38] Some of the other smart contract programming languages missing Turing-completeness are Simplicity, Scilla, Ivy and Bitcoin

Script.[38] However, measurements using regular expressions showed that only 35.3% of 53,757 Ethereum smart contracts included recursions and loops — constructs connected to the halting problem.[39]

Several languages are designed to enable formal verification: Bamboo, IELE, Simplicity, Michelson (can be verified with Coq),[38] Liquidity (compiles to Michelson), Scilla, DAML and Pact.[37]

Notable examples of blockchain platforms supporting smart contracts include the following:

Name	Description
Bitcoin	Provides a Turing-incomplete script language that allows the creation of custom smart contracts on top of Bitcoin like multisignature accounts, payment channels, escrows, time locks, atomic cross-chain trading, oracles, or multi-party lottery with no operator.[40]
Cardano	A blockchain platform for smart contracts, using proof of stake
Ethereum	Implements a Turing-complete language on its blockchain, a prominent smart contract framework[41]
EOS.IO	A blockchain platform for smart contracts
Tezos	A blockchain platform modifying its own set of rules with minimal disruption to the network through an on-chain governance model

Processes on a blockchain are generally deterministic in order to ensure Byzantine fault-tolerance.[42] Nevertheless, real world application of smart contracts, such as lotteries and casinos, require secure randomness.[43] In fact, blockchain technology reduces the costs for conducting of a lottery and is therefore beneficial for the participants. Randomness on blockchain can be implemented by using block hashes or timestamps, oracles, commitment schemes, special smart contracts like RANDAO[44][45] and Quanta as well as sequences from mixed strategy Nash equilibria.[42]

Applications

In 1998, Szabo proposed that smart contract infrastructure can be implemented by replicated asset registries and contract execution using cryptographic hash chains and Byzantine fault-tolerant replication.[46] Askemos implemented this approach in 2002[47][48] using Scheme (later adding SQLite[49][50]) as contract script language.[51]

One proposal for using bitcoin for replicated asset registration and contract execution is called "colored coins".[52] Replicated titles for potentially arbitrary forms of property, along with replicated contract execution, are implemented in different projects.

As of 2015, UBS was experimenting with "smart bonds" that use the bitcoin blockchain[53] in which payment streams could hypothetically be fully automated, creating a self-paying instrument.[54]

Inheritance wishes could hypothetically be implemented automatically upon registration of a death certificate by means of smart contracts.[55][56] Birth certificates can also work together with smart contracts.[57][58]

Chris Snook of *Inc.com* suggests smart contracts could also be used to handle real estate transactions and could be used in the field of title records and in the public register.[59][60][61][62][63]

Seth Oranburg and Liya Palagashvili argue that smart contracts could also be used in employment contracts, especially temporary employment contracts, which according to them would benefit the employer.[64][65]

Smart contract on block chain technology for smart villages[66]

Security issues

A blockchain-based smart contract is visible to all users of said blockchain. However, this leads to a situation where bugs, including security holes, are visible to all yet may not be quickly fixed.[67] Such an attack, difficult to fix quickly, was successfully executed on The DAO in June 2016, draining approximately US$50 million worth of Ether at the time, while developers attempted to come to a solution that would gain consensus.[68] The DAO program had a time delay in place before the hacker could remove the funds; a hard fork of the Ethereum software was done to claw back the funds from the attacker before the time limit expired.[69] Other high-profile attacks include the Parity multisignature wallet attacks, and an integer underflow/overflow attack (2018), totaling over US$184 million.[70]

Issues in Ethereum smart contracts, in particular, include ambiguities and easy-but-insecure constructs in its contract language Solidity, compiler bugs, Ethereum Virtual Machine bugs, attacks on the blockchain network, the immutability of bugs and that there is no central source documenting known vulnerabilities, attacks and problematic constructs.[41]

Difference from smart legal contracts

Smart legal contracts are distinct from smart contracts. As mentioned above, a smart contract is not necessarily legally enforceable as a contract. On the other hand, a smart legal contract has all the elements of a legally enforceable contract in the jurisdiction in which it can be enforced and it can be enforced by a court or tribunal. Therefore, while every smart legal contract will contain some elements of a smart contract, not every smart contract will be a smart legal contract.[71]

There is no formal definition of a smart legal contract in the legal industry.[72]

A Ricardian contract is a type of smart legal contract.

See also

- *Code and Other Laws of Cyberspace*
- Decentralized application
- Ethereum
- Regulation by algorithms
- Regulation of algorithms
- Ricardian contract (a design pattern to capture the intent of the agreement of parties)
- Loan
- Secure multiparty computation
- Transparency

References

1. Röscheisen, Martin; Baldonado, Michelle; Chang, Kevin; Gravano, Luis; Ketchpel, Steven; Paepcke, Andreas (1998). "The Stanford InfoBus and its service layers: Augmenting the internet with higher-level information management protocols". *Digital Libraries in Computer Science: The MeDoc Approach*. Lecture Notes in Computer Science. Springer. **1392**: 213–230. doi:10.1007/bfb0052526 (https://doi.org/10.1007%2Fbfb0052526). ISBN 978-3-540-64493-4.

2. Fries, Martin; P. Paal, Boris (2019). *Smart Contracts* (in German). Mohr Siebeck. ISBN 978-3-16-156911-1. JSTOR j.ctvn96h9r (https://www.jstor.org/stable/j.ctvn96h9r).
3. Savelyev, Alexander (14 December 2016). "Contract Law 2.0: "Smart" Contracts As the Beginning of the End of Classic Contract Law" (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2885241). Social Science Research Network. SSRN 2885241 (https://ssrn.com/abstract=2885241).
4. Tapscott, Don; Tapscott, Alex (May 2016). *The Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. pp. 72, 83, 101, 127. ISBN 978-0670069972.
5. Szabo, Nick (1997). "View of Formalizing and Securing Relationships on Public Networks | First Monday" (https://firstmonday.org/article/view/548/469). *First Monday*. doi:10.5210/fm.v2i9.548 (https://doi.org/10.5210%2Ffm.v2i9.548).
6. Zhou, Haozhe; Milani Fard, Amin; Makanju, Adetokunbo (2022-05-27). "The State of Ethereum Smart Contracts Security: Vulnerabilities, Countermeasures, and Tool Support" (https://doi.org/10.3390%2Fjcp2020019). *Journal of Cybersecurity and Privacy*. **2** (2): 358–378. doi:10.3390/jcp2020019 (https://doi.org/10.3390%2Fjcp2020019). ISSN 2624-800X (https://www.worldcat.org/issn/2624-800X).
7. "Crypto FAQ: What is a smart contract and how does it work?" (https://cryptocurrencyworks.com/crypto-faq/what-is-smart-contract.html). *Cryptocurrency Works*. Retrieved 2022-07-29.
8. "White Paper· ethereum/wiki Wiki · GitHub" (https://github.com/ethereum/wiki/wiki/White-Paper). *GitHub*. Archived (https://web.archive.org/web/20140111180823/http://ethereum.org/ethereum.html) from the original on 11 January 2014.
9. Alharby, Maher; van Moorsel, Aad (26 August 2017). "Blockchain-based Smart Contracts: A Systematic Mapping Study". *Computer Science & Information Technology*: 125–140. arXiv:1710.06372 (https://arxiv.org/abs/1710.06372). doi:10.5121/csit.2017.71011 (https://doi.org/10.5121%2Fcsit.2017.71011). ISBN 9781921987700. S2CID 725413 (https://api.semanticscholar.org/CorpusID:725413).
10. Cannarsa, Michel (1 December 2018). "Interpretation of Contracts and Smart Contracts: Smart Interpretation or Interpretation of Smart Contracts?" (https://kluwerlawonline.com/journalarticle/European+Review+of+Private+Law/26.6/ERPL2018054). *European Review of Private Law*. **26** (6): 773–785. doi:10.54648/ERPL2018054 (https://doi.org/10.54648%2FERPL2018054). S2CID 188017977 (https://api.semanticscholar.org/CorpusID:188017977).
11. Drummer, Daniel; Neumann, Dirk (5 August 2020). "Is code law? Current legal and technical adoption issues and remedies for blockchain-enabled smart contracts" (https://journals.sagepub.com/doi/abs/10.1177/0268396220924669). *Journal of Information Technology*. **35** (4): 337–360. doi:10.1177/0268396220924669 (https://doi.org/10.1177%2F0268396220924669). ISSN 0268-3962 (https://www.worldcat.org/issn/0268-3962). S2CID 225409384 (https://api.semanticscholar.org/CorpusID:225409384).
12. Filatova, Nataliia (1 September 2020). "Smart contracts from the contract law perspective: outlining new regulative strategies" (https://academic.oup.com/ijlit/article-abstract/28/3/217/5897086). *International Journal of Law and Information Technology*. **28** (3): 217–242. doi:10.1093/ijlit/eaaa015 (https://doi.org/10.1093%2Fijlit%2Feaaa015). ISSN 0967-0769 (https://www.worldcat.org/issn/0967-0769).
13. Morris, David Z. (21 January 2014). "Bitcoin is not just digital currency. It's Napster for finance" (http://fortune.com/2014/01/21/bitcoin-is-not-just-digital-currency-its-napster-for-finance/). *Fortune*. Retrieved 7 November 2018.
14. Schulpen, Ruben R.W.H.G. (1 August 2018). "Smart contracts in the Netherlands - University of Tilburg" (http://arno.uvt.nl/show.cgi?fid=146860). *uvt.nl*. Twente University. Retrieved 26 October 2019.
15. CleanApp (January 21, 2019). "Crypto's Founding Fallacy: How Mistakes in the 'Smart Contract' Genesis Block Weaken the Whole Chain" (https://medium.com/cryptolawreview/cryptos-founding-fallacy-aaa151b795ff). *Crypto Law Review*.

16. Mik, Eliza, Smart Contracts: A Requiem (December 7, 2019). Journal of Contract Law (2019) Volume 36 Part 1 at p 72
17. J Cieplak, S Leefatt, 'Smart Contracts: A Smart Way To Automate Performance' (2017) 1 Georgia L & Tech Rev 417
18. Governatori, Guido; Idelberger, Florian; Milosevic, Zoran; Riveret, Regis; Sartor, Giovanni; Xu, Xiwei (2018). "On legal contracts, imperative and declarative smart contracts, and blockchain systems". *Artificial Intelligence and Law*. **26** (4): 33. doi:10.1007/s10506-018-9223-3 (https://doi.org/10.1007%2Fs10506-018-9223-3). S2CID 3663005 (https://api.semanticscholar.org/CorpusID:3663005).
19. Buterin, Vitalik (August 7, 2015). "Ethereum - On Public and Public Blockchains" (https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/). *Ethereum.Org*.
20. D J Yaga et al., Blockchain Technology Overview, National Institute of Standards and Technology Internal/Interagency Report 8202, 2018, p 54, cited in Mik, Eliza, Smart Contracts: A Requiem (December 7, 2019). Journal of Contract Law (2019) Volume 36 Part 1 at p 71
21. Cachin, Christian. "Architecture of the Hyperledger Blockchain Fabric" (https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf) (PDF). *ibm.com*.
22. Vo, Hoang Tam; Kundu, Ashish; Mohania, Mukesh (2018). "Research Directions in Blockchain Data Management and Analytics" (https://openproceedings.org/2018/conf/edbt/paper-227.pdf) (PDF). *Advances in Database Technology - Extending Database Technology (EDBT)*. OpenProceedings. **21**: 446. "Some distributed ledger technologies support an additional capability called a smart contract, which is similar to the concept of stored procedure in classical relational databases to some extent. Smart contracts allow the shared business processes within a business network to be standardised, automated and enforced via computer programs to increase the integrity of the ledger."
23. Huckle, Steve; Bhattacharya, Rituparna; White, Martin; Beloff, Natalia (2016). "Internet of Things, Blockchain and Shared Economy Applications" (https://doi.org/10.1016%2Fj.procs.2016.09.074). *Procedia Computer Science*. Elsevier B.V. **98**: 463. doi:10.1016/j.procs.2016.09.074 (https://doi.org/10.1016%2Fj.procs.2016.09.074). "Firstly, that total quantity of BTC in a transaction's inputs must cover the total number of BTC in the outputs. That rule behaves similarly to a database stored procedure, except that it is impossible to circumvent. Secondly, BTC transactions use public-private key cryptography. That makes BTC act like a database with a publicly auditable per-row permission scheme."
24. Makhovsky, Andrei (December 22, 2017). "Belarus adopts crypto-currency law to woo foreign investors" (https://www.reuters.com/article/us-belarus-cryptocurrency-idUSKBN1EG0XO). *Reuters*.
25. Chapter 9: Building a Secure Future, One blockchain at a time (https://www.jec.senate.gov/public/_cache/files/aaac3a69-e9fb-45b6-be9f-b1fd96dd738b/chapter-9-building-a-secure-future-one-blockchain-at-a-time.pdf), US Senate Joint Economic Committee, March 2018.
26. "Arizona HB2417 - 2017 - Fifty-third Legislature 1st Regular" (https://legiscan.com/AZ/text/HB2417/id/1588180). *LegiScan*.
27. Hyman Gayle M, Digesti, Matthew P New Nevada legislation recognizes blockchain and smart contract terminologies (https://www.nvbar.org/wp-content/uploads/NevadaLawyer_Aug2017_Blockchain-1.pdf) August 2017, Nevada Lawyer
28. Tom, Daniel (22 September 2020). "Smart Contract Bill Tennessee" (http://www.capitol.tn.gov/Bills/110/Bill/SB1662.pdf) (PDF).
29. Wyoming, Legislature (26 February 2019). "Wyoming - Smart Contract" (https://wyoleg.gov/Legislation/2019/sf0125).
30. "Iowa House approves bills to facilitate broadband, cryptocurrency" (https://www.thegazette.com/government-politics/iowa-house-approves-bills-to-facilitate-broadband-cryptocurrency/). *www.thegazette.com*. Retrieved 2021-04-15.

31. Morgan, Herbert Smith Freehills LLP-Charlie; Parker, Chris; Livingston, Dorothy; Naish, Vanessa; Tevendale, Craig (23 April 2021). "Arbitration of digital disputes in smart contracts and the release of the digital dispute resolution rules from the UK jurisdiction taskforce | Lexology" (https://www.lexology.com/library/detail.aspx?g=6ea7c284-0157-4f2c-b330-e2758d1bf7a0). *www.lexology.com*. Retrieved 2021-04-25.
32. Soloro, Kevin; Kanna, Randall; Hoover, David (December 2019). *Hands-On Smart Contract Development With Solidity and Ethereum: From Fundamentals to Deployment* (https://www.google.com/books/edition/Hands_On_Smart_Contract_Development_with/thbADwAAQBAJ?hl=en&gbpv=1&kptab=getbook). California, U.S.A.: O'Reilly. p. 73. ISBN 978-1-492-04526-7. Retrieved 1 November 2020.
33. Sergey, Ilya; Nagaraj, Vaivaswatha; Johannsen, Jacob; Kumar, Amrit; Trunov, Anton; Hao, Ken Chan Guan (10 October 2019). "Safer smart contract programming with Scilla" (https://doi.org/10.1145%2F3360611). *Proceedings of the ACM on Programming Languages*. **3** (OOPSLA): 1–30. doi:10.1145/3360611 (https://doi.org/10.1145%2F3360611). ISSN 2475-1421 (https://www.worldcat.org/issn/2475-1421).
34. Alharby, Maher; van Moorsel, Aad (26 August 2017). "Blockchain-based Smart Contracts: A Systematic Mapping Study" (https://doi.org/10.5121%2Fcsit.2017.71011). *Computer Science & Information Technology (CS & IT)*: 125–140. arXiv:1710.06372 (https://arxiv.org/abs/1710.06372). doi:10.5121/csit.2017.71011 (https://doi.org/10.5121%2Fcsit.2017.71011). ISBN 9781921987700.
35. Wohrer, Maximilian; Zdun, Uwe (20 March 2018). "Smart contracts: security patterns in the ethereum ecosystem and solidity" (https://ieeexplore.ieee.org/document/8327565). *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*: 2–8. doi:10.1109/IWBOSE.2018.8327565 (https://doi.org/10.1109%2FIWBOSE.2018.8327565). ISBN 978-1-5386-5986-1. S2CID 4567923 (https://api.semanticscholar.org/CorpusID:4567923).
36. Perez, Daniel; Livshits, Benjamin (17 October 2020). "Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited". arXiv:1902.06710 (https://arxiv.org/abs/1902.06710) [cs.CR (https://arxiv.org/archive/cs.CR)].
37. Harz, Dominik; Knottenbelt, William (31 October 2018). "Towards Safer Smart Contracts: A Survey of Languages and Verification Methods". arXiv:1809.09805 (https://arxiv.org/abs/1809.09805) [cs.CR (https://arxiv.org/archive/cs.CR)].
38. Tyurin, A.V.; Tyuluandin, I.V.; Maltsev, V.S.; Kirilenko, I.A.; Berezun, D.A. (2019). "Overview of the Languages for Safe Smart Contract Programming" (https://doi.org/10.15514%2Fispras-2019-31%283%29-13). *Proceedings of the Institute for System Programming of the RAS*. **31** (3): 157–176. doi:10.15514/ispras-2019-31(3)-13 (https://doi.org/10.15514%2Fispras-2019-31%283%29-13). S2CID 203179644 (https://api.semanticscholar.org/CorpusID:203179644).
39. Jansen, Marc; Hdhili, Farouk; Gouiaa, Ramy; Qasem, Ziyaad (2020). "Do Smart Contract Languages Need to Be Turing Complete?". *Blockchain and Applications*. Advances in Intelligent Systems and Computing. Springer International Publishing. **1010**: 19–26. doi:10.1007/978-3-030-23813-1_3 (https://doi.org/10.1007%2F978-3-030-23813-1_3). ISBN 978-3-030-23812-4. S2CID 195656195 (https://api.semanticscholar.org/CorpusID:195656195). S2CID 195656195 (https://api.semanticscholar.org/CorpusID:195656195).
40. Atzei, Nicola; Bartoletti, Massimo; Cimoli, Tiziana; Lande, Stefano; Zunino, Roberto (2018), "SoK: unraveling Bitcoin smart contracts" (https://eprint.iacr.org/2018/192.pdf) (PDF), *7th International Conference on Principles of Security and Trust (POST)*, European Joint Conferences on Theory and Practice of Software
41. Atzei, Nicola; Bartoletti, Massimo; Cimoli, Tiziana (2017), "A survey of attacks on Ethereum smart contracts" (http://eprint.iacr.org/2016/1007.pdf) (PDF), *6th International Conference on Principles of Security and Trust (POST)*, European Joint Conferences on Theory and Practice of Software

42. Chatterjee, Krishnendu; Goharshady, Amir Kafshdar; Pourdamghani, Arash (21 February 2019). "Probabilistic Smart Contracts: Secure Randomness on the Blockchain". arXiv:1902.07986 (https://arxiv.org/abs/1902.07986) [cs.GT (https://arxiv.org/archive/cs.GT)].
43. Chen, Tai-yuan; Huang, Wei-ning; Kuo, Po-chun; Chung, Hao (6 August 2020). "Method for Generating Secure Randomness on Blockchain" (http://www.freepatentsonline.com/y2020/0252211.html). Retrieved 28 August 2020.
44. Jia, Zhifeng; Chen, Rui; Li, Jie (2019). "DeLottery: A Novel Decentralized Lottery System Based on Blockchain Technology". *Proceedings of the 2019 2nd International Conference on Blockchain Technology and Applications*. pp. 20–25. doi:10.1145/3376044.3376049 (https://doi.org/10.1145%2F3376044.3376049). ISBN 9781450377430. S2CID 207880557 (https://api.semanticscholar.org/CorpusID:207880557).
45. "randao/randao" (https://github.com/randao/randao). randao. 10 July 2020. Retrieved 10 July 2020.
46. Nick Szabo (1998). "Secure Property Titles with Owner Authority" (https://web.archive.org/web/20140115142013/http://szabo.best.vwh.net/securetitle.html). Archived from the original (http://szabo.best.vwh.net/securetitle.html) on January 15, 2014. Retrieved January 12, 2014.
47. Jörg F. Wittenberger (2002). *Askemos a distributed settlement* (http://citeseerx.ist.psu.edu/viewdoc/download;?doi=10.1.1.11.5050&rep=rep1&type=pdf). Proceedings of International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet (SSGRR), L'Aquila.
48. "Proceedings of International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet" (http://www.isl.cs.waseda.ac.jp/~sugawara/pdf/kurihara-SSGRR2002.pdf) (PDF).
49. Martin Möbius (2009). *Erstellung eines Archivierungskonzepts für die Speicherung rückverfolgbarer Datenbestände im Askemos-System* (https://monami.hs-mittweida.de/frontdoor/index/index/docId/476) (Thesis). Hochschule Mittweida.
50. Tom-Steve Watzke (2010). "Entwicklung einer Datenbankschnittstelle als Grundlage für Shop-Systeme unter dem Betriebssystem Askemos" (https://core.ac.uk/display/33987564).
51. RA Markus Heinker (2007). "Beweiswürdigung elektronischer Dokumente im Zivilprozess unter vergleichender Betrachtung von qualifizierten elektronischen Signaturen nach dem Signaturgesetz und dem Askemos-Verfahren" (http://askemos.org/A0e80fdd97a7b6e7af87c5d294f39a96c).
52. Hal Hodson (20 November 2013). "Bitcoin moves beyond mere money" (https://www.newscientist.com/article/dn24620-bitcoin-moves-beyond-mere-money.html). *New Scientist*. Retrieved 12 January 2014.
53. Ross, Rory (2015-09-12). "Smart Money: Blockchains Are the Future of the Internet" (http://europe.newsweek.com/smart-money-blockchains-are-future-internet-329278). *Newsweek*. Retrieved 2016-05-27.
54. Wigan, David (2015-06-11). "Bitcoin technology will disrupt derivatives, says banker" (http://www.ifrasia.com/bitcoin-technology-will-disrupt-derivatives-says-banker/21202956.article). *IFR Asia*. Retrieved 2016-05-27.
55. How blockchain technology could change our lives (https://www.europarl.europa.eu/RegData/etudes/IDAN/2017/581948/EPRS_IDA(2017)581948_EN.pdf)
56. Blockchain and AI are coming to kill these 4 business verticals (https://www.inc.com/chris-j-snook/4-small-business-verticals-artificial-intelligence-blockchain-will-destroy-in-coming-decade.html)
57. Blockchain for Digital Governments (https://publications.jrc.ec.europa.eu/repository/bitstream/JRC115049/blockchain_for_digital_government_online.pdf)
58. Blockchain Based Framework for Document Authentication (https://link.springer.com/chapter/10.1007/978-3-030-67490-8_19)

59. Snook, Chris J. (31 October 2017). "Blockchain and Artificial Intelligence Are Coming to Kill These 4 Small Business Verticals" (https://www.inc.com/chris-j-snook/4-small-business-verticals-artificial-intelligence-blockchain-will-destroy-in-coming-decade.html). *Inc.com*. Retrieved 25 January 2022.
60. The Bitfury Group and Government of Republic of Georgia Expand Blockchain Pilot (https://bitfury.com/content/downloads/the_bitfury_group_republic_of_georgia_expand_blockchain_pilot_2_7_16.pdf)
61. A BLOCKCHAIN - Journals Gateway (https://www.mitpressjournals.org/doi/pdf/10.1162/inov_a_00276)
62. Digital Transformation: Blockchain and Land Titles (https://www.oecd.org/corruption/integrity-forum/academic-papers/Georg%20Eder-%20Blockchain%20-%20Ghana_verified.pdf)
63. Ukraine launches big blockchain deal with tech firm Bitfury (https://www.reuters.com/article/us-ukraine-bitfury-blockchain-idUSKBN17F0N2)
64. Oranburg, Seth; Palagashvili, Liya (22 October 2018). "The Gig Economy, Smart Contracts, and Disruption of Traditional Work Arrangements" (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3270867). *Search eLibrary*. doi:10.2139/ssrn.3270867 (https://doi.org/10.2139%2Fssrn.3270867). S2CID 216803648 (https://api.semanticscholar.org/CorpusID:216803648). SSRN 3270867 (https://ssrn.com/abstract=3270867). Retrieved 25 January 2022.
65. A blockchain-based decentralized system for proper handling of temporary employment contracts (https://www.researchgate.net/publication/328657407_A_Blockchain-Based_Decentralized_System_for_Proper_Handling_of_Temporary_Employment_Contracts)
66. Kaur, Parminder; Parashar, Anshu (2022-06-01). "A Systematic Literature Review of Blockchain Technology for Smart Villages" (https://doi.org/10.1007/s11831-021-09659-7). *Archives of Computational Methods in Engineering*. **29** (4): 2417–2468. doi:10.1007/s11831-021-09659-7 (https://doi.org/10.1007%2Fs11831-021-09659-7). ISSN 1886-1784 (https://www.worldcat.org/issn/1886-1784). PMC 8549431 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8549431). PMID 34720578 (https://pubmed.ncbi.nlm.nih.gov/34720578).
67. Peck, M. (28 May 2016). "Ethereum's $150-Million Blockchain-Powered Fund Opens Just as Researchers Call For a Halt" (https://spectrum.ieee.org/tech-talk/computing/networks/ethereums-150-million-dollar-dao-opens-for-business-just-as-researchers-call-for-a-moratorium). *IEEE Spectrum*. Institute of Electrical and Electronics Engineers.
68. DuPont, Quinn (2017). "Experiments in Algorithmic Governance: A history and ethnography of "The DAO", a failed Decentralized Autonomous Organization" (https://web.archive.org/web/20170730133911/http://iqdupont.com/assets/documents/DUPONT%2D2017%2DPreprint%2DAlgorithmic%2DGovernance.pdf) (PDF). Archived from the original (http://iqdupont.com/assets/documents/DUPONT-2017-Preprint-Algorithmic-Governance.pdf) (PDF) on 2017-07-30. Retrieved 29 July 2017.
69. Coy, Peter; Kharif, Olga (25 August 2016). "This Is Your Company on Blockchain" (https://www.bloomberg.com/news/articles/2016-08-25/this-is-your-company-on-blockchain). *Bloomberg Businessweek*. Retrieved 2016-12-05.
70. Praitheeshan, Purathani; Pan, Lei; Yu, Jiangshan; Liu, Joseph; Doss, R. (2019). "Security Analysis Methods on Ethereum Smart Contract Vulnerabilities: A Survey". arXiv:1908.08605 (https://arxiv.org/abs/1908.08605) [cs.CR (https://arxiv.org/archive/cs.CR)].
71. Whitepaper: Smart Contracts and Distributed Ledger – A Legal Perspective (https://www.isda.org/a/6EKDE/smart-contracts-and-distributed-ledger-a-legal-perspective.pdf), 5.
72. Whitepaper: Smart Contracts and Distributed Ledger – A Legal Perspective (https://www.isda.org/a/6EKDE/smart-contracts-and-distributed-ledger-a-legal-perspective.pdf), 3.

Turing completeness

In computability theory, a system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is said to be **Turing-complete** or **computationally universal** if it can be used to simulate any Turing machine (devised by English mathematician and computer scientist Alan Turing). This means that this system is able to recognize or decide other data-manipulation rule sets. Turing completeness is used as a way to express the power of such a data-manipulation rule set. Virtually all programming languages today are Turing-complete.

A related concept is that of **Turing equivalence** – two computers P and Q are called equivalent if P can simulate Q and Q can simulate P. The Church–Turing thesis conjectures that any function whose values can be computed by an algorithm can be computed by a Turing machine, and therefore that if any real-world computer can simulate a Turing machine, it is Turing equivalent to a Turing machine. A universal Turing machine can be used to simulate any Turing machine and by extension the computational aspects of any possible real-world computer.[NB 1]

To show that something is Turing-complete, it is enough to show that it can be used to simulate some Turing-complete system. No physical system can have infinite memory, but if the limitation of finite memory is ignored, most programming languages are otherwise Turing-complete.

Contents

Non-mathematical usage

In colloquial usage, the terms "Turing-complete" and "Turing-equivalent" are used to mean that any real-world general-purpose computer or computer language can approximately simulate the computational aspects of any other real-world general-purpose computer or computer language. In

real life this leads to the practical concepts of computing virtualization and emulation.

Real computers constructed so far can be functionally analyzed like a single-tape Turing machine (the "tape" corresponding to their memory); thus the associated mathematics can apply by abstracting their operation far enough. However, real computers have limited physical resources, so they are only linear bounded automaton complete. In contrast, a universal computer is defined as a device with a Turing-complete instruction set, infinite memory, and infinite available time.

Formal definitions

In computability theory, several closely related terms are used to describe the computational power of a computational system (such as an abstract machine or programming language):

Turing completeness
: A computational system that can compute every Turing-computable function is called Turing-complete (or Turing-powerful). Alternatively, such a system is one that can simulate a universal Turing machine.

Turing equivalence
: A Turing-complete system is called Turing-equivalent if every function it can compute is also Turing-computable; i.e., it computes precisely the same class of functions as do Turing machines. Alternatively, a Turing-equivalent system is one that can simulate, and be simulated by, a universal Turing machine. (All known physically-implementable Turing-complete systems are Turing-equivalent, which adds support to the Church–Turing thesis.)

(Computational) universality
: A system is called universal with respect to a class of systems if it can compute every function computable by systems in that class (or can simulate each of those systems). Typically, the term 'universality' is tacitly used with respect to a Turing-complete class of systems. The term "weakly universal" is sometimes used to distinguish a system (e.g. a cellular automaton) whose universality is achieved only by modifying the standard definition of Turing machine so as to include input streams with infinitely many 1s.

History

Turing completeness is significant in that every real-world design for a computing device can be simulated by a universal Turing machine. The Church–Turing thesis states that this is a law of mathematics – that a universal Turing machine can, in principle, perform any calculation that any other programmable computer can. This says nothing about the effort needed to write the program, or the time it may take for the machine to perform the calculation, or any abilities the machine may possess that have nothing to do with computation.

Charles Babbage's analytical engine (1830s) would have been the first Turing-complete machine if it had been built at the time it was designed. Babbage appreciated that the machine was capable of great feats of calculation, including primitive logical reasoning, but he did not appreciate that no other machine could do better. From the 1830s until the 1940s, mechanical calculating machines such as adders and multipliers were built and improved, but they could not perform a conditional branch and therefore were not Turing-complete.

In the late 19th century, Leopold Kronecker formulated notions of computability, defining primitive recursive functions. These functions can be calculated by rote computation, but they are not enough to make a universal computer, because the instructions that compute them do not allow for an infinite loop. In the early 20th century, David Hilbert led a program to axiomatize all of mathematics with precise axioms and precise logical rules of deduction that could be performed by a machine. Soon it became clear that a small set of deduction rules are enough to produce the consequences of any set of axioms. These rules were proved by Kurt Gödel in 1930 to be enough to produce every theorem.

The actual notion of computation was isolated soon after, starting with Gödel's incompleteness theorem. This theorem showed that axiom systems were limited when reasoning about the computation that deduces their theorems. Church and Turing independently demonstrated that Hilbert's *Entscheidungsproblem* (decision problem) was unsolvable,[1] thus identifying the computational core of the incompleteness theorem. This work, along with Gödel's work on general recursive functions, established that there are sets of simple instructions, which, when put together, are able to produce any computation. The work of Gödel showed that the notion of computation is essentially unique.

In 1941 Konrad Zuse completed the Z3 computer. Zuse was not familiar with Turing's work on computability at the time. In particular, the Z3 lacked dedicated facilities for a conditional jump, thereby precluding it from being Turing complete. However, in 1998, it was shown by Rojas that the Z3 is capable of simulating conditional jumps, and therefore Turing complete in theory. To do this, its tape program would have to be long enough to execute every possible path through both sides of every branch.[2]

The first computer capable of conditional branching in practice, and therefore Turing complete in practice, was the ENIAC in 1946. Zuse's Z4 computer was operational in 1945, but it did not support conditional branching until 1950.[3]

Computability theory

Computability theory uses models of computation to analyze problems and determine whether they are computable and under what circumstances. The first result of computability theory is that there exist problems for which it is impossible to predict what a (Turing-complete) system will do over an arbitrarily long time.

The classic example is the halting problem: create an algorithm that takes as input a program in some Turing-complete language and some data to be fed to *that* program, and determines whether the program, operating on the input, will eventually stop or will continue forever. It is trivial to create an algorithm that can do this for *some* inputs, but impossible to do this in general. For any characteristic of the program's eventual output, it is impossible to determine whether this characteristic will hold.

This impossibility poses problems when analyzing real-world computer programs. For example, one cannot write a tool that entirely protects programmers from writing infinite loops or protects users from supplying input that would cause infinite loops.

One can instead limit a program to executing only for a fixed period of time (timeout) or limit the power of flow-control instructions (for example, providing only loops that iterate over the items of an existing array). However, another theorem shows that there are problems solvable by Turing-complete languages that cannot be solved by any language with only finite looping abilities (i.e., any language guaranteeing that every program will eventually finish to a halt). So any such language is not Turing-complete. For example, a language in which programs are guaranteed to complete and halt cannot compute the computable function produced by Cantor's diagonal argument on all computable functions in that language.

Turing oracles

A computer with access to an infinite tape of data may be more powerful than a Turing machine: for instance, the tape might contain the solution to the halting problem or some other Turing-undecidable problem. Such an infinite tape of data is called a Turing oracle. Even a Turing oracle

with random data is not computable (with probability 1), since there are only countably many computations but uncountably many oracles. So a computer with a random Turing oracle can compute things that a Turing machine cannot.

Digital physics

All known laws of physics have consequences that are computable by a series of approximations on a digital computer. A hypothesis called digital physics states that this is no accident because the universe itself is computable on a universal Turing machine. This would imply that no computer more powerful than a universal Turing machine can be built physically.

Examples

The computational systems (algebras, calculi) that are discussed as Turing-complete systems are those intended for studying theoretical computer science. They are intended to be as simple as possible, so that it would be easier to understand the limits of computation. Here are a few:

- Automata theory
- Formal grammar (language generators)
- Formal language (language recognizers)
- Lambda calculus
- Post–Turing machines
- Process calculus

Most programming languages (their abstract models, maybe with some particular constructs that assume finite memory omitted), conventional and unconventional, are Turing-complete. This includes:

- All general-purpose languages in wide use.
  - Procedural programming languages such as C, Pascal.
  - Object-oriented languages such as Java, Smalltalk or C#.
  - Multi-paradigm languages such as Ada, C++, Common Lisp, Fortran, JavaScript, Object Pascal, Perl, Python, R.
- Most languages using less common paradigms:
  - Functional languages such as Lisp and Haskell.
  - Logic programming languages such as Prolog.
  - General-purpose macro processor such as m4.
  - Declarative languages such as XSLT.[4]
  - VHDL and other hardware description languages.
  - TeX, a typesetting system.
  - Esoteric programming languages, a form of mathematical recreation in which programmers work out how to achieve basic programming constructs in an extremely difficult but mathematically Turing-equivalent language.

Some rewrite systems are Turing-complete.

Turing completeness is an abstract statement of ability, rather than a prescription of specific language features used to implement that ability. The features used to achieve Turing completeness can be quite different; Fortran systems would use loop constructs or possibly even goto statements to achieve repetition; Haskell and Prolog, lacking looping almost entirely, would use recursion.

Most programming languages are describing computations on von Neumann architectures, which have memory (RAM and register) and a control unit. These two elements make this architecture Turing-complete. Even pure functional languages are Turing-complete.[5][NB 2]

Turing completeness in declarative SQL is implemented through recursive common table expressions.[6] Unsurprisingly, procedural extensions to SQL (PLSQL, etc.) are also Turing-complete. This illustrates one reason why relatively powerful non-Turing-complete languages are rare: the more powerful the language is initially, the more complex are the tasks to which it is applied and the sooner its lack of completeness becomes perceived as a drawback, encouraging its extension until it is Turing-complete.

The untyped lambda calculus is Turing-complete, but many typed lambda calculi, including System F, are not. The value of typed systems is based in their ability to represent most typical computer programs while detecting more errors.

Rule 110 and Conway's Game of Life, both cellular automata, are Turing-complete.

Unintentional Turing completeness

Some games and other software are Turing-complete by accident, i.e. not by design.

Software:

- Microsoft Excel[7]
- Microsoft PowerPoint[8]

Video games:

- *Dwarf Fortress*[9]
- *Cities: Skylines*[10]
- *Opus Magnum*[11]
- Minecraft[12]

Social media:

- *Habbo Hotel*[13]

Computational languages:

- C++ templates[14]
- printf format string[15]
- Python type hints[16]

Computer hardware:

- x86 MOV instruction[17]

Biology:

- Chemical reaction networks[18][19][20][21] and enzyme-based DNA computers[22] have been shown to be Turing-equivalent

Non-Turing-complete languages

Many computational languages exist that are not Turing-complete. One such example is the set of regular languages, which are generated by regular expressions and which are recognized by finite automata. A more powerful but still not Turing-complete extension of finite automata is the category of pushdown automata and context-free grammars, which are commonly used to generate parse trees in an initial stage of program compiling. Further examples include some of the early versions of the pixel shader languages embedded in Direct3D and OpenGL extensions.

In total functional programming languages, such as Charity and Epigram, all functions are total and must terminate. Charity uses a type system and control constructs based on category theory, whereas Epigram uses dependent types. The LOOP language is designed so that it computes only the functions that are primitive recursive. All of these compute proper subsets of the total computable functions, since the full set of total computable functions is not computably enumerable. Also, since all functions in these languages are total, algorithms for recursively enumerable sets cannot be written in these languages, in contrast with Turing machines.

Although (untyped) lambda calculus is Turing-complete, simply typed lambda calculus is not.

See also

- AI-completeness
- Algorithmic information theory
- Chomsky hierarchy
- Church–Turing thesis
- Computability theory
- Inner loop
- Loop (computing)
- Machine that always halts
- Rice's theorem
- s_{mn} theorem
- Structured program theorem
- Turing tarpit
- Virtualization
- Emulation (computing)

Notes

1. A UTM cannot simulate non-computational aspects such as I/O.
2. The following book provides an introduction for computation models: Rauber, Thomas; Rünger, Gudula (2013). *Parallel programming: for multicore and cluster systems* (https://books.google.com/books?id=UbpAAAAAQBAJ) (2nd ed.). Springer. ISBN 9783642378010.

References

1. Hodges, Andrew (1992) [1983], *Alan Turing: The Enigma*, London: Burnett Books, p. 111, ISBN 0-04-510060-8
2. Rojas, Raul (1998). "How to make Zuse's Z3 a universal computer" (https://www.researchgate.net/publication/3330654). *Annals of the History of Computing*. **20** (3): 51–54. doi:10.1109/85.707574 (https://doi.org/10.1109%2F85.707574).
3. Rojas, Raúl (1 February 2014). Google translation (https://translate.google.com/translate?sl=auto&tl=en&js=y&prev=_t&hl=en&ie=UTF-8&u=https%3A%2F%2Fwww.mi.fu-berlin.de%2Finf%2Fgroups/ag-ki%2Fpublications%2FBedingter-Sprung%2F&edit-text=), pdf is also translatable. "Konrad Zuse und der bedingte Sprung" [Konrad Zuse and the conditional jump]. *Informatik-Spektrum* (in German). **37** (1): 50–53. doi:10.1007/s00287-013-0717-9 (https://doi.org/10.1007%2Fs00287-013-0717-9). ISSN 0170-6012 (https://www.worldcat.org/issn/0170-6012). S2CID 1086397 (https://api.semanticscholar.org/CorpusID:1086397). {{cite journal}}: External link in |others= (help)

4. Lyons, Bob (30 March 2001). "Universal Turing Machine in XSLT" (http://www.unidex.com/turing/utm.htm). *B2B Integration Solutions from Unidex*. Archived (https://web.archive.org/web/20110717162708/http://www.unidex.com/turing/utm.htm) from the original on 17 July 2011. Retrieved 5 July 2010.
5. Boyer, Robert S.; Moore, J. Strother (May 1983). *A Mechanical Proof of the Turing Completeness of Pure Lisp* (http://www.cs.utexas.edu/users/boyer/ftp/ics-reports/cmp37.pdf) (PDF) (Technical report). Institute for Computing Science, University of Texas at Austin. 37. Archived (https://web.archive.org/web/20170922044624/http://www.cs.utexas.edu/users/boyer/ftp/ics-reports/cmp37.pdf) (PDF) from the original on 22 September 2017.
6. Dfetter; Breinbaas (8 August 2011). "Cyclic Tag System" (https://wiki.postgresql.org/index.php?title=Cyclic_Tag_System&oldid=15106). *PostgreSQL wiki*. Retrieved 10 September 2014.
7. "Announcing LAMBDA: Turn Excel formulas into custom functions" (https://techcommunity.microsoft.com/t5/excel-blog/announcing-lambda-turn-excel-formulas-into-custom-functions/ba-p/1925546). *TECHCOMMUNITY.MICROSOFT.COM*. 3 December 2020. Retrieved 8 December 2020.
8. Wildenhain, Tom (9 April 2020). "On Turing Completeness of MS PowerPoint" (https://www.andrew.cmu.edu/user/twildenh/PowerPointTM/Paper.pdf) (PDF).
9. Cedotal, Andrew (16 April 2010). "Man Uses World's Most Difficult Computer Game to Create … A Working Turing Machine" (http://www.themarysue.com/dwarf-fortress-turing-machine-computer/). *The Mary Sue*. Archived (https://web.archive.org/web/20150627102458/http://www.themarysue.com/dwarf-fortress-turing-machine-computer/) from the original on 27 June 2015. Retrieved 2 June 2015.
10. Plunkett, Luke (16 July 2019). "Cities: Skylines Map Becomes A Poop-Powered Computer" (https://kotaku.com/cities-skylines-map-becomes-a-poop-powered-calculator-1836398063). *Kotaku*. Retrieved 16 July 2019.
11. Caldwell, Brendan (20 November 2017). "Opus Magnum player makes an alchemical computer" (https://www.rockpapershotgun.com/2017/11/20/opus-magnum-player-makes-computer/). *Rock Paper Shotgun*. Retrieved 23 September 2019.
12. Writer, Staff. "This 8-bit processor built in Minecraft can run its own games" (https://www.pcworld.com/article/559794/8-bit-computer-processor-built-in-minecraft-can-run-its-own-games.html). *PCWorld*. Retrieved 21 July 2022.
13. "Habbo's Twitter thread on the implementation of a Turing machine inside the game" (https://twitter.com/Habbo/status/1325785673304043522). 9 November 2020. Retrieved 11 November 2020.
14. Meyers, Scott (Scott Douglas) (2005). *Effective C++ : 55 specific ways to improve your programs and designs* (https://archive.org/details/effectivec55spec00meye) (3rd ed.). Upper Saddle River, NJ: Addison-Wesley. ISBN 0321334876. OCLC 60425273 (https://www.worldcat.org/oclc/60425273).
15. A 27th IOCCC Winner (https://www.ioccc.org/2020/carlini/index.html)
Carlini, Nicolas; Barresi, Antonio; Payer, Mathias; Wagner, David; Gross, Thomas R. (August 2015). "Control-flow bending: on the effectiveness of control-flow integrity" (https://dl.acm.org/doi/10.5555/2831143.2831154). *Proceedings of the 24th USENIX Conference on Security Symposium*. pp. 161–176. ISBN 9781931971232.
16. Roth, Ori (31 August 2022). "Python Type Hints are Turing Complete" (https://arxiv.org/abs/2208.14755).
17. Dolan, Stephen. "mov is Turing-complete" (https://web.archive.org/web/20210214020524/https://stedolan.net/research/mov.pdf) (PDF). *stedolan.net*. Archived from the original (http://stedolan.net/research/mov.pdf) (PDF) on 14 February 2021. Retrieved 9 May 2019.

18. Shah, Shalin; Wee, Jasmine; Song, Tianqi; Ceze, Luis; Strauss, Karin; Chen, Yuan-Jyue; Reif, John (4 May 2020). "Using Strand Displacing Polymerase To Program Chemical Reaction Networks". *Journal of the American Chemical Society*. **142** (21): 9587–9593. doi:10.1021/jacs.0c02240 (https://doi.org/10.1021%2Fjacs.0c02240). ISSN 0002-7863 (https://www.worldcat.org/issn/0002-7863). PMID 32364723 (https://pubmed.ncbi.nlm.nih.gov/32364723). S2CID 218504535 (https://api.semanticscholar.org/CorpusID:218504535).
19. Chen, Yuan-Jyue; Dalchau, Neil; Srinivas, Niranjan; Phillips, Andrew; Cardelli, Luca; Soloveichik, David; Seelig, Georg (October 2013). "Programmable chemical controllers made from DNA" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4150546). *Nature Nanotechnology*. **8** (10): 755–762. Bibcode:2013NatNa...8..755C (https://ui.adsabs.harvard.edu/abs/2013NatNa...8..755C). doi:10.1038/nnano.2013.189 (https://doi.org/10.1038%2Fnnano.2013.189). ISSN 1748-3395 (https://www.worldcat.org/issn/1748-3395). PMC 4150546 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4150546). PMID 24077029 (https://pubmed.ncbi.nlm.nih.gov/24077029).
20. Srinivas, Niranjan; Parkin, James; Seelig, Georg; Winfree, Erik; Soloveichik, David (15 December 2017). "Enzyme-free nucleic acid dynamical systems" (https://doi.org/10.1126%2Fscience.aal2052). *Science*. **358** (6369): eaal2052. doi:10.1126/science.aal2052 (https://doi.org/10.1126%2Fscience.aal2052). ISSN 0036-8075 (https://www.worldcat.org/issn/0036-8075). PMID 29242317 (https://pubmed.ncbi.nlm.nih.gov/29242317).
21. Soloveichik, David; Seelig, Georg; Winfree, Erik (23 March 2010). "DNA as a universal substrate for chemical kinetics" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2851759). *Proceedings of the National Academy of Sciences*. **107** (12): 5393–5398. Bibcode:2010PNAS..107.5393S (https://ui.adsabs.harvard.edu/abs/2010PNAS..107.5393S). doi:10.1073/pnas.0909380107 (https://doi.org/10.1073%2Fpnas.0909380107). ISSN 0027-8424 (https://www.worldcat.org/issn/0027-8424). PMC 2851759 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2851759). PMID 20203007 (https://pubmed.ncbi.nlm.nih.gov/20203007).
22. Shapiro, Ehud (7 December 1999). "A Mechanical Turing Machine: Blueprint for a Biomolecular Computer" (https://web.archive.org/web/20090103224150/http://www.wisdom.weizmann.ac.il/~udi/DNA5/scripps_short/index.htm). *Interface Focus*. Weizmann Institute of Science. **2** (4): 497–503. doi:10.1098/rsfs.2011.0118 (https://doi.org/10.1098%2Frsfs.2011.0118). PMC 3363030 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3363030). PMID 22649583 (https://pubmed.ncbi.nlm.nih.gov/22649583). Archived from the original (http://www.wisdom.weizmann.ac.il/~udi/DNA5/scripps_short/index.htm) on 3 January 2009. Retrieved 13 August 2009.

Further reading

- Brainerd, W. S.; Landweber, L. H. (1974). *Theory of Computation* (https://archive.org/details/theoryofcomputat00brai). Wiley. ISBN 0-471-09585-0.
- Giles, Jim (24 October 2007). "Simplest 'universal computer' wins student $25,000" (https://www.newscientist.com/article/dn12826-simplest-universal-computer-wins-student-25000.html). *New Scientist*.
- Herken, Rolf, ed. (1995). *The Universal Turing Machine: A Half-Century Survey*. Springer Verlag. ISBN 3-211-82637-8.
- Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem" (https://www.cs.ox.ac.uk/activities/ieg/e-library/sources/tp2-ie.pdf) (PDF). *Proceedings of the London Mathematical Society*. 2. **42**: 230–265. doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230). S2CID 73712 (https://api.semanticscholar.org/CorpusID:73712).
- Turing, A. M. (1938). "On Computable Numbers, with an Application to the Entscheidungsproblem: A correction". *Proceedings of the London Mathematical Society*. 2. **43**: 544–546. doi:10.1112/plms/s2-43.6.544 (https://doi.org/10.1112%2Fplms%2Fs2-43.6.544).

External links

- "Turing Complete" (https://c2.com/cgi/wiki?TuringComplete). *wiki.c2.com*.

Church–Turing thesis

In computability theory, the **Church–Turing thesis** (also known as **computability thesis**,[1] the **Turing–Church thesis**,[2] the **Church–Turing conjecture**, **Church's thesis**, **Church's conjecture**, and **Turing's thesis**) is a thesis about the nature of computable functions. It states that a function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine. The thesis is named after American mathematician Alonzo Church and the British mathematician Alan Turing. Before the precise definition of computable function, mathematicians often used the informal term effectively calculable to describe functions that are computable by paper-and-pencil methods. In the 1930s, several independent attempts were made to formalize the notion of computability:

- In 1933, Kurt Gödel, with Jacques Herbrand, formalized the definition of the class of general recursive functions: the smallest class of functions (with arbitrarily many arguments) that is closed under composition, recursion, and minimization, and includes zero, successor, and all projections.
- In 1936, Alonzo Church created a method for defining functions called the λ-calculus. Within λ-calculus, he defined an encoding of the natural numbers called the Church numerals. A function on the natural numbers is called λ-computable if the corresponding function on the Church numerals can be represented by a term of the λ-calculus.
- Also in 1936, before learning of Church's work,[6] Alan Turing created a theoretical model for machines, now called Turing machines, that could carry out calculations from inputs by manipulating symbols on a tape. Given a suitable encoding of the natural numbers as sequences of symbols, a function on the natural numbers is called Turing computable if some Turing machine computes the corresponding function on encoded natural numbers.

Church,[7] Kleene,[8] and Turing[9][11] proved that these three formally defined classes of computable functions coincide: a function is λ-computable if and only if it is Turing computable, and if and only if it is *general recursive*. This has led mathematicians and computer scientists to believe that the concept of computability is accurately characterized by these three equivalent processes. Other formal attempts to characterize computability have subsequently strengthened this belief (see below).

On the other hand, the Church–Turing thesis states that the above three formally-defined classes of computable functions coincide with the *informal* notion of an effectively calculable function. Although the thesis has near-universal acceptance, it cannot be formally proven, as the concept of effective calculability is only informally defined.

Since its inception, variations on the original thesis have arisen, including statements about what can physically be realized by a computer in our universe (physical Church-Turing thesis) and what can be efficiently computed (Church–Turing thesis (complexity theory)). These variations are not due to Church or Turing, but arise from later work in complexity theory and digital physics. The thesis also has implications for the philosophy of mind (see below).

Contents

Statement in Church's and Turing's words

J. B. Rosser (1939) addresses the notion of "effective computability" as follows: "Clearly the existence of CC and RC (Church's and Rosser's proofs) presupposes a precise definition of 'effective'. 'Effective method' is here used in the rather special sense of a method each step of which is precisely predetermined and which is certain to produce the answer in a finite number of steps".[12] Thus the adverb-adjective "effective" is used in a sense of "1a: producing a decided, decisive, or desired effect", and "capable of producing a result".[13][14]

In the following, the words "effectively calculable" will mean "produced by any intuitively 'effective' means whatsoever" and "effectively computable" will mean "produced by a Turing-machine or equivalent mechanical device". Turing's "definitions" given in a footnote in his 1938 Ph.D. thesis *Systems of Logic Based on Ordinals*, supervised by Church, are virtually the same:

> † We shall use the expression "computable function" to mean a function calculable by a machine, and let "effectively calculable" refer to the intuitive idea without particular identification with any one of these definitions.[15]

The thesis can be stated as: *Every effectively calculable function is a computable function*.[16] Church also stated that "No computational procedure will be considered as an algorithm unless it can be represented as a Turing Machine". Turing stated it this way:

> It was stated ... that "a function is effectively calculable if its values can be found by some purely mechanical process". We may take this literally, understanding that by a purely mechanical process one which could be carried out by a machine. The development ... leads to ... an identification of computability † with effective calculability. [† is the footnote quoted above.][15]

History

One of the important problems for logicians in the 1930s was the Entscheidungsproblem of David Hilbert and Wilhelm Ackermann,[17] which asked whether there was a mechanical procedure for separating mathematical truths from mathematical falsehoods. This quest required that the notion of "algorithm" or "effective calculability" be pinned down, at least well enough for the quest to

begin.[18] But from the very outset Alonzo Church's attempts began with a debate that continues to this day.[19] Was the notion of "effective calculability" to be (i) an "axiom or axioms" in an axiomatic system, (ii) merely a *definition* that "identified" two or more propositions, (iii) an *empirical hypothesis* to be verified by observation of natural events, or (iv) just *a proposal* for the sake of argument (i.e. a "thesis").

Circa 1930–1952

In the course of studying the problem, Church and his student Stephen Kleene introduced the notion of λ-definable functions, and they were able to prove that several large classes of functions frequently encountered in number theory were λ-definable.[20] The debate began when Church proposed to Gödel that one should define the "effectively computable" functions as the λ-definable functions. Gödel, however, was not convinced and called the proposal "thoroughly unsatisfactory".[21] Rather, in correspondence with Church (c. 1934–1935), Gödel proposed *axiomatizing* the notion of "effective calculability"; indeed, in a 1935 letter to Kleene, Church reported that:

> His [Gödel's] only idea at the time was that it might be possible, in terms of effective calculability as an undefined notion, to state a set of axioms which would embody the generally accepted properties of this notion, and to do something on that basis.[22]

But Gödel offered no further guidance. Eventually, he would suggest his recursion, modified by Herbrand's suggestion, that Gödel had detailed in his 1934 lectures in Princeton NJ (Kleene and Rosser transcribed the notes). But he did not think that the two ideas could be satisfactorily identified "except heuristically".[23]

Next, it was necessary to identify and prove the equivalence of two notions of effective calculability. Equipped with the λ-calculus and "general" recursion, Stephen Kleene with help of Church and J. Barkley Rosser produced proofs (1933, 1935) to show that the two calculi are equivalent. Church subsequently modified his methods to include use of Herbrand–Gödel recursion and then proved (1936) that the Entscheidungsproblem is unsolvable: there is no algorithm that can determine whether a well formed formula has a beta normal form.[24]

Many years later in a letter to Davis (c. 1965), Gödel said that "he was, at the time of these [1934] lectures, not at all convinced that his concept of recursion comprised all possible recursions".[25] By 1963–1964 Gödel would disavow Herbrand–Gödel recursion and the λ-calculus in favor of the Turing machine as the definition of "algorithm" or "mechanical procedure" or "formal system".[26]

A hypothesis leading to a natural law?: In late 1936 Alan Turing's paper (also proving that the Entscheidungsproblem is unsolvable) was delivered orally, but had not yet appeared in print.[27] On the other hand, Emil Post's 1936 paper had appeared and was certified independent of Turing's work.[28] Post strongly disagreed with Church's "identification" of effective computability with the λ-calculus and recursion, stating:

> Actually the work already done by Church and others carries this identification considerably beyond the working hypothesis stage. But to mask this identification under a definition... blinds us to the need of its continual verification.[29]

Rather, he regarded the notion of "effective calculability" as merely a "working hypothesis" that might lead by inductive reasoning to a "natural law" rather than by "a definition or an axiom".[30] This idea was "sharply" criticized by Church.[31]

Thus Post in his 1936 paper was also discounting Kurt Gödel's suggestion to Church in 1934–1935 that the thesis might be expressed as an axiom or set of axioms.[22]

Turing adds another definition, Rosser equates all three: Within just a short time, Turing's 1936–1937 paper "On Computable Numbers, with an Application to the Entscheidungsproblem"[27] appeared. In it he stated another notion of "effective computability" with the introduction of his a-machines (now known as the Turing machine abstract computational model). And in a proof-sketch added as an "Appendix" to his 1936–1937 paper, Turing showed that the classes of functions defined by λ-calculus and Turing machines coincided.[32] Church was quick to recognise how compelling Turing's analysis was. In his review of Turing's paper he made clear that Turing's notion made "the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately".[33]

In a few years (1939) Turing would propose, like Church and Kleene before him, that *his* formal definition of mechanical computing agent was the correct one.[34] Thus, by 1939, both Church (1934) and Turing (1939) had individually proposed that their "formal systems" should be *definitions* of "effective calculability";[35] neither framed their statements as *theses*.

Rosser (1939) formally identified the three notions-as-definitions:

> All three *definitions* are equivalent, so it does not matter which one is used.[36]

Kleene proposes *Thesis I*: This left the overt expression of a "thesis" to Kleene. In 1943 Kleene proposed his "Thesis I":[37]

> This heuristic fact [general recursive functions are effectively calculable] ... led Church to state the following thesis. The same thesis is implicit in Turing's description of computing machines.
>
> > Thesis I. *Every effectively calculable function (effectively decidable predicate) is general recursive* [Kleene's italics]
>
> Since a precise mathematical definition of the term effectively calculable (effectively decidable) has been wanting, we can take this thesis ... as a definition of it ...
>
> ...the thesis has the character of an hypothesis—a point emphasized by Post and by Church. If we consider the thesis and its converse as definition, then the hypothesis is an hypothesis about the application of the mathematical theory developed from the definition. For the acceptance of the hypothesis, there are, as we have suggested, quite compelling grounds.

The Church–Turing Thesis: Stephen Kleene, in *Introduction To Metamathematics*, finally goes on to formally name "Church's Thesis" and "Turing's Thesis", using his theory of recursive realizability. Kleene having switched from presenting his work in the terminology of Church-Kleene lambda definability, to that of Gödel-Kleene recursiveness (partial recursive functions). In

this transition, Kleene modified Gödel's general recursive functions to allow for proofs of the unsolvability of problems in the Intuitionism of E. J. Brouwer. In his graduate textbook on logic, "Church's thesis" is introduced and basic mathematical results are demonstrated to be unrealizable. Next, Kleene proceeds to present "Turing's thesis", where results are shown to be uncomputable, using his simplified derivation of a Turing machine based on the work of Emil Post. Both theses are proven equivalent by use of "Theorem XXX".

> Thesis I. *Every effectively calculable function (effectively decidable predicate) is general recursive.*[38]

> Theorem XXX: The following classes of partial functions are coextensive, i.e. have the same members: (a) the partial recursive functions, (b) the computable functions ...[39]

> Turing's thesis: Turing's thesis that every function which would naturally be regarded as computable is computable under his definition, i.e. by one of his machines, is equivalent to Church's thesis by Theorem XXX.[39]

Kleene, finally, uses for the first time the term the "Church-Turing thesis" in a section in which he helps to give clarifications to concepts in Alan Turing's paper "The Word Problem in Semi-Groups with Cancellation", as demanded in a critique from William Boone.[40]

Later developments

An attempt to understand the notion of "effective computability" better led Robin Gandy (Turing's student and friend) in 1980 to analyze *machine* computation (as opposed to human-computation acted out by a Turing machine). Gandy's curiosity about, and analysis of, cellular automata (including Conway's game of life), parallelism, and crystalline automata, led him to propose four "principles (or constraints) ... which it is argued, any machine must satisfy".[41] His most-important fourth, "the principle of causality" is based on the "finite velocity of propagation of effects and signals; contemporary physics rejects the possibility of instantaneous action at a distance".[42] From these principles and some additional constraints—(1a) a lower bound on the linear dimensions of any of the parts, (1b) an upper bound on speed of propagation (the velocity of light), (2) discrete progress of the machine, and (3) deterministic behavior—he produces a theorem that "What can be calculated by a device satisfying principles I–IV is computable."[43]

In the late 1990s Wilfried Sieg analyzed Turing's and Gandy's notions of "effective calculability" with the intent of "sharpening the informal notion, formulating its general features axiomatically, and investigating the axiomatic framework".[44] In his 1997 and 2002 work Sieg presents a series of constraints on the behavior of a *computor*—"a human computing agent who proceeds mechanically". These constraints reduce to:

- "(B.1) (Boundedness) There is a fixed bound on the number of symbolic configurations a computor can immediately recognize.
- "(B.2) (Boundedness) There is a fixed bound on the number of internal states a computor can be in.
- "(L.1) (Locality) A computor can change only elements of an observed symbolic configuration.
- "(L.2) (Locality) A computor can shift attention from one symbolic configuration to another one, but the new observed configurations must be within a bounded distance of the immediately previously observed configuration.

- "(D) (Determinacy) The immediately recognizable (sub-)configuration determines uniquely the next computation step (and id [instantaneous description])"; stated another way: "A computor's internal state together with the observed configuration fixes uniquely the next computation step and the next internal state."[45]

The matter remains in active discussion within the academic community.[46][47]

The thesis as a definition

The thesis can be viewed as nothing but an ordinary mathematical definition. Comments by Gödel on the subject suggest this view, e.g. "the correct definition of mechanical computability was established beyond any doubt by Turing".[48] The case for viewing the thesis as nothing more than a definition is made explicitly by Robert I. Soare,[5] where it is also argued that Turing's definition of computability is no less likely to be correct than the epsilon-delta definition of a continuous function.

Success of the thesis

Other formalisms (besides recursion, the λ-calculus, and the Turing machine) have been proposed for describing effective calculability/computability. Stephen Kleene (1952) adds to the list the functions "*reckonable* in the system S_1" of Kurt Gödel 1936, and Emil Post's (1943, 1946) "*canonical* [also called *normal*] *systems*".[49] In the 1950s Hao Wang and Martin Davis greatly simplified the one-tape Turing-machine model (see Post–Turing machine). Marvin Minsky expanded the model to two or more tapes and greatly simplified the tapes into "up-down counters", which Melzak and Lambek further evolved into what is now known as the counter machine model. In the late 1960s and early 1970s researchers expanded the counter machine model into the register machine, a close cousin to the modern notion of the computer. Other models include combinatory logic and Markov algorithms. Gurevich adds the pointer machine model of Kolmogorov and Uspensky (1953, 1958): "... they just wanted to ... convince themselves that there is no way to extend the notion of computable function."[50]

All these contributions involve proofs that the models are computationally equivalent to the Turing machine; such models are said to be Turing complete. Because all these different attempts at formalizing the concept of "effective calculability/computability" have yielded equivalent results, it is now generally assumed that the Church–Turing thesis is correct. In fact, Gödel (1936) proposed something stronger than this; he observed that there was something "absolute" about the concept of "reckonable in S_1":

> It may also be shown that a function which is computable ['reckonable'] in one of the systems S_i, or even in a system of transfinite type, is already computable [reckonable] in S_1. Thus the concept 'computable' ['reckonable'] is in a certain definite sense 'absolute', while practically all other familiar metamathematical concepts (e.g. provable, definable, etc.) depend quite essentially on the system to which they are defined ...[51]

Informal usage in proofs

Proofs in computability theory often invoke the Church–Turing thesis in an informal way to establish the computability of functions while avoiding the (often very long) details which would be involved in a rigorous, formal proof.[52] To establish that a function is computable by Turing

machine, it is usually considered sufficient to give an informal English description of how the function can be effectively computed, and then conclude "by the Church–Turing thesis" that the function is Turing computable (equivalently, partial recursive).

Dirk van Dalen gives the following example for the sake of illustrating this informal use of the Church–Turing thesis:[53]

> Example: Each infinite RE set contains an infinite recursive set.
>
> Proof: Let A be infinite RE. We list the elements of A effectively, n_0, n_1, n_2, n_3, ...
>
> From this list we extract an increasing sublist: put $m_0 = n_0$, after finitely many steps we find an n_k such that $n_k > m_0$, put $m_1 = n_k$. We repeat this procedure to find $m_2 > m_1$, etc. this yields an effective listing of the subset $B=\{m_0, m_1, m_2,...\}$ of A, with the property $m_i < m_{i+1}$.
>
> *Claim*. B is decidable. For, in order to test k in B we must check if $k = m_i$ for some i. Since the sequence of m_i's is increasing we have to produce at most k+1 elements of the list and compare them with k. If none of them is equal to k, then k not in B. Since this test is effective, B is decidable and, **by Church's thesis**, recursive.

In order to make the above example completely rigorous, one would have to carefully construct a Turing machine, or λ-function, or carefully invoke recursion axioms, or at best, cleverly invoke various theorems of computability theory. But because the computability theorist believes that Turing computability correctly captures what can be computed effectively, and because an effective procedure is spelled out in English for deciding the set B, the computability theorist accepts this as proof that the set is indeed recursive.

Variations

The success of the Church–Turing thesis prompted variations of the thesis to be proposed. For example, the **physical Church–Turing thesis** states: "All physically computable functions are Turing-computable."[54]:101

The Church–Turing thesis says nothing about the efficiency with which one model of computation can simulate another. It has been proved for instance that a (multi-tape) universal Turing machine only suffers a logarithmic slowdown factor in simulating any Turing machine.[55]

A variation of the Church–Turing thesis addresses whether an arbitrary but "reasonable" model of computation can be efficiently simulated. This is called the **feasibility thesis**,[56] also known as the (**classical**) **complexity-theoretic Church–Turing thesis** or the **extended Church–Turing thesis**, which is not due to Church or Turing, but rather was realized gradually in the development of complexity theory. It states:[57] "A probabilistic Turing machine can efficiently simulate any realistic model of computation." The word 'efficiently' here means up to polynomial-time reductions. This thesis was originally called **computational complexity-theoretic Church–Turing thesis** by Ethan Bernstein and Umesh Vazirani (1997). The complexity-theoretic Church–Turing thesis, then, posits that all 'reasonable' models of computation yield the same class of problems that can be computed in polynomial time. Assuming the conjecture that probabilistic polynomial time (BPP) equals deterministic polynomial time (P), the word 'probabilistic' is optional in the complexity-theoretic Church–Turing thesis. A similar thesis, called the **invariance thesis**, was introduced by Cees F. Slot and Peter van Emde Boas. It states: "'Reasonable' machines can simulate each other within a polynomially bounded overhead in time

and a constant-factor overhead in space."[58] The thesis originally appeared in a paper at STOC'84, which was the first paper to show that polynomial-time overhead and constant-space overhead could be *simultaneously* achieved for a simulation of a Random Access Machine on a Turing machine.[59]

If BQP is shown to be a strict superset of BPP, it would invalidate the complexity-theoretic Church–Turing thesis. In other words, there would be efficient quantum algorithms that perform tasks that do not have efficient probabilistic algorithms. This would not however invalidate the original Church–Turing thesis, since a quantum computer can always be simulated by a Turing machine, but it would invalidate the classical complexity-theoretic Church–Turing thesis for efficiency reasons. Consequently, the **quantum complexity-theoretic Church–Turing thesis** states:[57] "A quantum Turing machine can efficiently simulate any realistic model of computation."

Eugene Eberbach and Peter Wegner claim that the Church–Turing thesis is sometimes interpreted too broadly, stating "Though [...] Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid".[60] They claim that forms of computation not captured by the thesis are relevant today, terms which they call super-Turing computation.

Philosophical implications

Philosophers have interpreted the Church–Turing thesis as having implications for the philosophy of mind.[61][62][63] B. Jack Copeland states that it is an open empirical question whether there are actual deterministic physical processes that, in the long run, elude simulation by a Turing machine; furthermore, he states that it is an open empirical question whether any such processes are involved in the working of the human brain.[64] There are also some important open questions which cover the relationship between the Church–Turing thesis and physics, and the possibility of hypercomputation. When applied to physics, the thesis has several possible meanings:

1. The universe is equivalent to a Turing machine; thus, computing non-recursive functions is physically impossible. This has been termed the strong Church–Turing thesis, or Church–Turing–Deutsch principle, and is a foundation of digital physics.
2. The universe is not equivalent to a Turing machine (i.e., the laws of physics are not Turing-computable), but incomputable physical events are not "harnessable" for the construction of a hypercomputer. For example, a universe in which physics involves random real numbers, as opposed to computable reals, would fall into this category.
3. The universe is a hypercomputer, and it is possible to build physical devices to harness this property and calculate non-recursive functions. For example, it is an open question whether all quantum mechanical events are Turing-computable, although it is known that rigorous models such as quantum Turing machines are equivalent to deterministic Turing machines. (They are not necessarily efficiently equivalent; see above.) John Lucas and Roger Penrose have suggested that the human mind might be the result of some kind of quantum-mechanically enhanced, "non-algorithmic" computation.[65][66]

There are many other technical possibilities which fall outside or between these three categories, but these serve to illustrate the range of the concept.

Philosophical aspects of the thesis, regarding both physical and biological computers, are also discussed in Odifreddi's 1989 textbook on recursion theory.[67]:101–123

Non-computable functions

One can formally define functions that are not computable. A well-known example of such a function is the Busy Beaver function. This function takes an input *n* and returns the largest number of symbols that a Turing machine with *n* states can print before halting, when run with no input. Finding an upper bound on the busy beaver function is equivalent to solving the halting problem, a problem known to be unsolvable by Turing machines. Since the busy beaver function cannot be computed by Turing machines, the Church–Turing thesis states that this function cannot be effectively computed by any method.

Several computational models allow for the computation of (Church-Turing) non-computable functions. These are known as hypercomputers. Mark Burgin argues that super-recursive algorithms such as inductive Turing machines disprove the Church–Turing thesis.[68] His argument relies on a definition of algorithm broader than the ordinary one, so that non-computable functions obtained from some inductive Turing machines are called computable. This interpretation of the Church–Turing thesis differs from the interpretation commonly accepted in computability theory, discussed above. The argument that super-recursive algorithms are indeed algorithms in the sense of the Church–Turing thesis has not found broad acceptance within the computability research community.

See also

- Abstract machine
- Church's thesis in constructive mathematics
- Church–Turing–Deutsch principle, which states that every physical process can be simulated by a universal computing device
- Computability logic
- Computability theory
- Decidability
- Hypercomputation
- Model of computation
- Oracle (computer science)
- Super-recursive algorithm
- Turing completeness

Footnotes

1. Robert Soare, "Turing Oracle Machines, Online Computing, and Three Displacements in Computability Theory" (http://www.people.cs.uchicago.edu/~soare/History/turing.pdf)
2. Rabin, Michael O. (June 2012). *Turing, Church, Gödel, Computability, Complexity and Randomization: A Personal View* (http://videolectures.net/turing100_rabin_turing_church_goedel/). Event occurs at 9:36. Retrieved 2021-12-05.
3. Correspondence between Max Newman and Church in Alonzo Church papers (https://findingaids.princeton.edu/collections/C0948/c00385)
4. Turing, Alan (2004). *The essential Turing : seminal writings in computing, logic, philosophy, artificial intelligence, and artificial life, plus the secrets of Enigma* (http://www.cse.chalmers.se/~aikmitr/papers/Turing.pdf) (PDF). Oxford: Clarendon Press. p. 44. ISBN 9780198250791. Retrieved 2021-12-06.
5. Soare, Robert I. (September 1996). "Computability and Recursion". *Bulletin of Symbolic Logic*. **2** (3): 284–321. CiteSeerX 10.1.1.35.5803 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.5803). doi:10.2307/420992 (https://doi.org/10.2307%2F420992). JSTOR 420992 (https://www.jstor.org/stable/420992).

6. Church's paper was presented to the American Mathematical Society on 19 April 1935 and published on 15 April 1936. Turing, who had made substantial progress in writing up his own results, was disappointed to learn of Church's proof upon its publication.[3][4] Turing quickly completed his paper and rushed it to publication; it was received by the *Proceedings of the London Mathematical Society* on 28 May 1936, read on 12 November 1936, and published in series 2, volume 42 (1936–1937); it appeared in two sections: in Part 3 (pages 230–240), issued on 30 November 1936 and in Part 4 (pages 241–265), issued on 23 December 1936; Turing added corrections in volume 43 (1937), pp. 544–546.[5]:45
7. Church 1936
8. Kleene 1936
9. Turing 1937a
10. Kleene 1936
11. Turing 1937b. Proof outline on p. 153: $\lambda\text{-definable} \overset{triv}{\Longrightarrow} \lambda\text{-}K\text{-definable} \overset{160}{\Longrightarrow} \text{Turing computable} \overset{161}{\Longrightarrow} \mu\text{-recursive} \overset{Kleene}{\Longrightarrow}$ [10] λ-definable
12. Rosser 1939 in Davis 1965:225.
13. "effective". *Merriam Webster's New Collegiate Dictionary* (9th ed.).
14. See also "effective". *Merriam-Webster's Online Dictionary* (http://www.merriam-webster.com/dictionary/effective) (11th ed.). Retrieved 2014-07-26, which also gives these definitions for "effective" – the first ["producing a decided, decisive, or desired effect"] as the definition for sense "1a" of the word "effective", and the second ["capable of producing a result"] as part of the "Synonym Discussion of EFFECTIVE" there, (in the introductory part, where it summarizes the similarities between the meanings of the words "effective", "effectual", "efficient", and "efficacious").
15. Turing, A. M. (1938). *Systems of Logic Based on Ordinals* (https://web.archive.org/web/20121023103503/https://webspace.princeton.edu/users/jedwards/Turing%20Centennial%202012/Mudd%20Archive%20files/12285_AC100_Turing_1938.pdf) (PDF) (PhD). Princeton University. p. 8. Archived from the original (https://webspace.princeton.edu/users/jedwards/Turing%20Centennial%202012/Mudd%20Archive%20files/12285_AC100_Turing_1938.pdf) (PDF) on 2012-10-23. Retrieved 2012-06-23.
16. Gandy (1980:123) states it this way: *What is effectively calculable is computable.* He calls this "Church's Thesis".
17. David Hilbert and Wilhelm Ackermann: Grundzüge der theoretischen Logik, Berlin, Germany, Springer, 1st ed. 1928. (6th ed. 1972, ISBN 3-540-05843-5) English Translation: David Hilbert and Wilhelm Ackermann: Principles of Mathematical Logic, AMS Chelsea Publishing, Providence, Rhode Island, USA, 1950.
18. Davis's commentary before Church 1936 *An Unsolvable Problem of Elementary Number Theory* in Davis 1965:88. Church uses the words "effective calculability" on page 100ff.
19. In his review of *Church's Thesis after 70 Years* edited by Adam Olszewski et al. 2006, Peter Smith's criticism of a paper by Muraswski and Wolenski suggests 4 "lines" re the status of the Church–Turing Thesis: (1) empirical hypothesis (2) axiom or theorem, (3) definition, (4) explication. But Smith opines that (4) is indistinguishable from (3), cf. Smith (2007-07-11) *Church's Thesis after 70 Years* at http://www.logicmatters.net/resources/pdfs/CTT.pdf
20. cf. footnote 3 in Church 1936a *An Unsolvable Problem of Elementary Number Theory* in Davis 1965:89.
21. Dawson 1997:99.
22. Sieg 1997:160.
23. Sieg 1997:160 quoting from the 1935 letter written by Church to Kleene, cf. Footnote 3 in Gödel 1934 in Davis 1965:44.
24. cf. Church 1936 in Davis 1965:105ff..
25. Davis's commentary before Gödel 1934 in Davis 1965:40.

26. For a detailed discussion of Gödel's adoption of Turing's machines as models of computation, see Shagrir. "Goedel on Turing on Computability" (https://web.archive.org/web/20151217145831/http://moon.cc.huji.ac.il/oron-shagrir/papers/Goedel_on_Turing_on_Computability.pdf) (PDF). Archived from the original (http://moon.cc.huji.ac.il/oron-shagrir/papers/Goedel_on_Turing_on_Computability.pdf) (PDF) on 2015-12-17. Retrieved 2016-02-08.
27. Turing 1937.
28. cf. Editor's footnote to Post 1936 *Finite Combinatory Process. Formulation I.* at Davis 1965:289.
29. Post 1936 in Davis 1965:291, footnote 8.
30. Post 1936 in Davis 1952:291.
31. Sieg 1997:171 and 176–177.
32. Turing 1936–1937 in Davis 1965:263ff..
33. Church 1937.
34. Turing 1939 in Davis:160.
35. cf. Church 1934 in Davis 1965:100, also Turing 1939 in Davis 1965:160.
36. italics added, Rosser 1939 in Davis 1965:226.
37. Kleene 1943, p. 60 in Davis 1965:274. Footnotes omitted.
38. Kleene 1952:300.
39. Kleene 1952:376.
40. Kleene 1952:382, 536
41. Gandy 1980:123ff.
42. Gandy 1980:135
43. Gandy 1980:126
44. Sieg 1998–1999 in Sieg, Somner & Talcott 2002:390ff.; also Sieg 1997:154ff.
45. In a footnote Sieg breaks Post's 1936 (B) into (B.1) and (B.2) and (L) into (L.1) and (L.2) and describes (D) differently. With respect to his proposed Gandy machine he later adds LC.1, LC.2, GA.1 and GA.2. These are complicated; see Sieg 1998–1999 in Sieg, Somner & Talcott 2002:390ff..
46. A collection of papers can be found in Olszewski, Woleński & Janusz (2006). Also a review of this collection: Smith, Peter (2007-07-11). "Church's Thesis after 70 Years" (http://www.logicmatters.net/resources/pdfs/CTT.pdf) (PDF).
47. See also Hodges, Andrew (2005). "Did Church and Turing Have a Thesis about Machines?" (https://web.archive.org/web/20160304032827/http://www.turing.org.uk/publications/ct70.pdf) (PDF). Archived from the original (http://www.turing.org.uk/publications/ct70.pdf) (PDF) on 2016-03-04. Retrieved 2014-07-27.
48. Gödel, Kurt (1995) [193?]. "Undecidable Diophantine Propositions" (https://books.google.com/books?id=gDzbuUwma5MC&pg=PA164). In Feferman, Solomon (ed.). *Collected Works* (https://books.google.com/books?id=gDzbuUwma5MC). Vol. 3. New York: Oxford University Press. p. 168 (https://books.google.com/books?id=gDzbuUwma5MC&pg=PA168). ISBN 978-0-19-507255-6. OCLC 928791907 (https://www.worldcat.org/oclc/928791907).
49. Kleene 1952:320
50. Gurevich 1988:2
51. Translation of Gödel (1936) by Davis in *The Undecidable* p. 83, differing in the use of the word 'reckonable' in the translation in Kleene (1952) p. 321
52. Horsten in Olszewski 2006:256.
53. Gabbay 2001:284

54. Piccinini, Gualtiero (January 2007). "Computationalism, the Church–Turing Thesis, and the Church–Turing Fallacy" (http://www.umsl.edu/~piccininig/Computationalism_Church-Turing_Thesis_Church-Turing_Fallacy.pdf) (PDF). *Synthese*. **154** (1): 97–120. CiteSeerX 10.1.1.360.9796 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.360.9796). doi:10.1007/s11229-005-0194-z (https://doi.org/10.1007%2Fs11229-005-0194-z). S2CID 494161 (https://api.semanticscholar.org/CorpusID:494161).
55. Arora, Sanjeev; Barak, Boaz (2009). *Complexity Theory: A Modern Approach* (http://www.cs.princeton.edu/theory/complexity/). Cambridge University Press. ISBN 978-0-521-42426-4. Sections 1.4, "Machines as strings and the universal Turing machine" and 1.7, "Proof of theorem 1.9".
56. "Official Problem Description" (https://web.archive.org/web/20051124084833/http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf) (PDF). Archived from the original (http://www.claymath.org/millennium/P_vs_NP/Official_Problem_Description.pdf) (PDF) on 2005-11-24.
57. Kaye, Phillip; Laflamme, Raymond; Mosca, Michele (2007). *An introduction to quantum computing*. Oxford University Press. pp. 5–6. ISBN 978-0-19-857049-3.
58. van Emde Boas, Peter (1990). "Machine Models and Simulations". *Handbook of Theoretical Computer Science A*. Elsevier. p. 5.
59. Slot, C.; van Emde Boas, P. (December 1984). *On tape versus core: an application of space efficient perfect hash functions to the invariance of space*. STOC.
60. Eberbach & Wegner 2003, p. 287.
61. Abramson, Darren (2011). "Philosophy of mind is (in part) philosophy of computer science" (https://dl.acm.org/doi/abs/10.1007/s11023-011-9236-0). *Minds and Machines*. **21** (2): 203–219. doi:10.1007/s11023-011-9236-0 (https://doi.org/10.1007%2Fs11023-011-9236-0). S2CID 32116031 (https://api.semanticscholar.org/CorpusID:32116031).
62. Copeland, B. Jack (2017-11-10). "The Church-Turing Thesis" (https://plato.stanford.edu/entries/church-turing/). In Zalta, Edward N. (ed.). *Stanford Encyclopedia of Philosophy*.
63. For a good place to encounter original papers see Chalmers, David J., ed. (2002). *Philosophy of Mind: Classical and Contemporary Readings*. New York: Oxford University Press. ISBN 978-0-19-514581-6. OCLC 610918145 (https://www.worldcat.org/oclc/610918145).
64. Copeland, B. Jack (2004). "Computation". In Floridi, Luciano (ed.). *The Blackwell guide to the philosophy of computing and information*. Wiley-Blackwell. p. 15. ISBN 978-0-631-22919-3.
65. cf. Penrose, Roger (1990). "Algorithms and Turing machines". *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford: Oxford University Press. pp. 47–49. ISBN 978-0-19-851973-7. OCLC 456785846 (https://www.worldcat.org/oclc/456785846).
66. Also the description of "the non-algorithmic nature of mathematical insight", Penrose, Roger (1990). "Where lies the physics of mind?". *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford: Oxford University Press. pp. 416–418. ISBN 978-0-19-851973-7. OCLC 456785846 (https://www.worldcat.org/oclc/456785846).
67. Piergiorgio Odifreddi (1989). *Classical Recursion Theory*. Studies in Logic and the Foundations of Mathematics. Vol. 125. Amsterdam, Netherlands: North Holland.
68. Burgin, Mark (2005). *Super-Recursive Algorithms*. Monographs in Computer Science. New York: Springer. ISBN 978-0-387-95569-8. OCLC 990755791 (https://www.worldcat.org/oclc/990755791).

References

- Barwise, Jon; Keisler, H.J.; Kunen, Kenneth, eds. (1980). *The Kleene Symposium*. Amsterdam: North-Holland Publishing Company. ISBN 978-0-444-85345-5.
- Ben-Amram, A. M. (2005). "The Church-Turing Thesis and its Look-Alikes" (http://www2.mta.ac.il/~amirben/downloadable/church-turing.ps) (PS). *SIGACT News*. **36** (3): 113–116.

CiteSeerX 10.1.1.74.7308 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.7308). doi:10.1145/1086649.1086651 (https://doi.org/10.1145%2F1086649.1086651). S2CID 13566703 (https://api.semanticscholar.org/CorpusID:13566703).

- Bernstein, E.; Vazirani, U. (1997). "Quantum complexity theory". *SIAM Journal on Computing*. **26** (5): 1411–1473. CiteSeerX 10.1.1.655.1186 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.655.1186). doi:10.1137/S0097539796300921 (https://doi.org/10.1137%2FS0097539796300921).
- Blass, Andreas; Gurevich, Yuri (October 2003). "Algorithms: A Quest for Absolute Definitions" (http://research.microsoft.com/~gurevich/Opera/164.pdf) (PDF). *Bulletin of European Association for Theoretical Computer Science* (81).
- Burgin, Mark (2005). "Super-recursive algorithms". *Monographs in computer science*. Springer. ISBN 978-0-387-95569-8.
- Church, Alonzo (1932). "A set of Postulates for the Foundation of Logic". *Annals of Mathematics*. **33** (2): 346–366. doi:10.2307/1968337 (https://doi.org/10.2307%2F1968337). JSTOR 1968337 (https://www.jstor.org/stable/1968337).
- Church, Alonzo (April 1936a). "An Unsolvable Problem of Elementary Number Theory" (https://web.archive.org/web/20200227091349/https://pdfs.semanticscholar.org/ee95/8b752cdfc62c16289cd8d3b7274a2e09b14e.pdf) (PDF). *American Journal of Mathematics*. **58** (2): 345–363. doi:10.2307/2371045 (https://doi.org/10.2307%2F2371045). JSTOR 2371045 (https://www.jstor.org/stable/2371045). Archived from the original (https://pdfs.semanticscholar.org/ee95/8b752cdfc62c16289cd8d3b7274a2e09b14e.pdf) (PDF) on 2020-02-27.
- Church, Alonzo (June 1936b). "A Note on the Entscheidungsproblem". *Journal of Symbolic Logic*. **1** (1): 40–41. doi:10.2307/2269326 (https://doi.org/10.2307%2F2269326). JSTOR 2269326 (https://www.jstor.org/stable/2269326).
- Church, Alonzo (March 1937). "Review: A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem". *Journal of Symbolic Logic*. **2** (1): 42–43. doi:10.2307/2268810 (https://doi.org/10.2307%2F2268810). JSTOR 2268810 (https://www.jstor.org/stable/2268810).
- Church, Alonzo (1941). *The Calculi of Lambda-Conversion*. Princeton: Princeton University Press.
- Cooper, S. B.; Odifreddi, P. (2003). "Incomputability in Nature". In S. B. Cooper; S. S. Goncharov (eds.). *Computability and Models: Perspectives East and West*. Kluwer Academic/Plenum Publishers. pp. 137–160.
- Davis, Martin, ed. (1965). *The Undecidable, Basic Papers on Undecidable Propositions, Unsolvable Problems And Computable Functions* (https://archive.org/details/undecidablebasic0000davi). New York: Raven Press. Includes original papers by Gödel, Church, Turing, Rosser, Kleene, and Post mentioned in this section.
- Dawson, John W. Jr. (1997). *Logical Dilemmas: The Life and Work of Kurt Gödel*. Wellesley, Massachusetts, USA: A. K. Peters.
- Eberbach, E.; Wegner, P. (October 2003). "Beyond Turing Machines" (http://eatcs.org/images/bulletin/beatcs81.pdf#page=287) (PDF). *Bulletin of the European Association for Theoretical Computer Science* (81): 279–304. CiteSeerX 10.1.1.61.9759 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.9759).
- Gabbay, D. M. (2001). *Handbook of Philosophical Logic*. Vol. 1 (2nd ed.).
- Gandy, Robin (1980). "Church's Thesis and the Principles for Mechanisms". In H. J. Barwise; H. J. Keisler; K. Kunen (eds.). *The Kleene Symposium*. North-Holland Publishing Company. pp. 123–148.
- Gandy, Robin (1994). Herken, Rolf (ed.). *The universal Turing Machine: A Half-Century Survey*. New York: Wien Springer–Verlag. pp. 51ff. ISBN 978-3-211-82637-9.
- Gödel, Kurt (1965) [1934]. "On Undecidable Propositions of Formal Mathematical Systems". In Davis, Martin (ed.). *The Undecidable* (https://archive.org/details/undecidablebasic0000davi).

Kleene and Rosser (lecture note-takers); Institute for Advanced Study (lecture sponsor). New York: Raven Press.

- Gödel, Kurt (1936). "Über die Lāange von Beweisen" [On The Length of Proofs]. *Ergenbnisse Eines Mathematishen Kolloquiums* (in German). Heft (7): 23–24. Cited by Kleene (1952).
- Gurevich, Yuri (June 1988). "On Kolmogorov Machines and Related Issues". *Bulletin of European Association for Theoretical Computer Science* (35): 71–82.
- Gurevich, Yuri (July 2000). "Sequential Abstract State Machines Capture Sequential Algorithms" (http://research.microsoft.com/~gurevich/Opera/141.pdf) (PDF). *ACM Transactions on Computational Logic*. **1** (1): 77–111. CiteSeerX 10.1.1.146.3017 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.146.3017). doi:10.1145/343369.343384 (https://doi.org/10.1145%2F343369.343384). S2CID 2031696 (https://api.semanticscholar.org/CorpusID:2031696).
- Herbrand, Jacques (1932). "Sur la non-contradiction de l'arithmétique". *Journal für die Reine und Angewandte Mathematik* (in French). **166**: 1–8.
- Hofstadter, Douglas R. "Chapter XVII: Church, Turing, Tarski, and Others". *Gödel, Escher, Bach: an Eternal Golden Braid*.
- Kleene, Stephen Cole (January 1935). "A Theory of Positive Integers in Formal Logic". *American Journal of Mathematics*. **57** (1): 153–173 & 219–244. doi:10.2307/2372027 (https://doi.org/10.2307%2F2372027). JSTOR 2372027 (https://www.jstor.org/stable/2372027).
- Kleene, Stephen Cole (1936). "Lambda-Definability and Recursiveness". *Duke Mathematical Journal*. **2** (2): 340–353. doi:10.1215/s0012-7094-36-00227-2 (https://doi.org/10.1215%2Fs0012-7094-36-00227-2).
- Kleene, Stephen Cole (1943). "Recursive Predicates and Quantifiers" (https://doi.org/10.2307%2F1990131). *Transactions of the American Mathematical Society*. **53** (1): 41–73. doi:10.2307/1990131 (https://doi.org/10.2307%2F1990131). JSTOR 1990131 (https://www.jstor.org/stable/1990131). Reprinted in *The Undecidable*, p. 255ff. Kleene refined his definition of "general recursion" and proceeded in his chapter "12. Algorithmic theories" to posit "Thesis I" (p. 274); he would later repeat this thesis (in Kleene 1952:300) and name it "Church's Thesis" (Kleene 1952:317) (i.e., the Church thesis).
- Kleene, Stephen Cole (1952). *Introduction to Metamathematics*. North-Holland. OCLC 523942 (https://www.worldcat.org/oclc/523942).
- Knuth, Donald (1973). *The Art of Computer Programming*. Vol. 1/Fundamental Algorithms (2nd ed.). Addison–Wesley.
- Kugel, Peter (November 2005). "It's time to think outside the computational box". *Communications of the ACM*. **48** (11): 32–37. CiteSeerX 10.1.1.137.6939 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.137.6939). doi:10.1145/1096000.1096001 (https://doi.org/10.1145%2F1096000.1096001). S2CID 29843806 (https://api.semanticscholar.org/CorpusID:29843806).
- Lewis, H.R.; Papadimitriou, C.H. (1998). *Elements of the Theory of Computation*. Upper Saddle River, New Jersey, USA: Prentice-Hall.
- Manna, Zohar (2003) [1974]. *Mathematical Theory of Computation*. Dover. ISBN 978-0-486-43238-0.
- Markov, A. A. (1960) [1954]. "The Theory of Algorithms". *American Mathematical Society Translations*. **2** (15): 1–14.
- Olszewski, Adam; Woleński, Jan; Janusz, Robert, eds. (2006). *Church's Thesis After 70 Years*. Frankfurt: Ontos. ISBN 978-3-938793-09-1. OCLC 909679288 (https://www.worldcat.org/oclc/909679288).
- Pour-El, M. B.; Richards, J.I. (1989). *Computability in Analysis and Physics*. Springer Verlag.
- Rosser, J. B. (1939). "An Informal Exposition of Proofs of Godel's Theorem and Church's Theorem". *The Journal of Symbolic Logic*. **4** (2): 53–60. doi:10.2307/2269059 (https://doi.org/10.2307%2F2269059). JSTOR 2269059 (https://www.jstor.org/stable/2269059).
- Sieg, Wilfried; Sommer, Richard; Talcott, Carolyn, eds. (2002). *Reflections on the Foundations of Mathematics: Essays in Honor of Solomon Feferman*. Lecture Notes in Logic. Vol. 15. A. K.

Peters, Ltd. ISBN 978-1-56881-169-7.
- Syropoulos, Apostolos (2008). *Hypercomputation: Computing Beyond the Church–Turing Barrier*. Springer. ISBN 978-0-387-30886-9.
- Turing, A. M. (1937) [Delivered to the Society November 1936], "On Computable Numbers, with an Application to the Entscheidungsproblem" (http://www.comlab.ox.ac.uk/activities/ieg/e-library/sources/tp2-ie.pdf) (PDF), *Proceedings of the London Mathematical Society*, 2, vol. 42, pp. 230–265, doi:10.1112/plms/s2-42.1.230 (https://doi.org/10.1112%2Fplms%2Fs2-42.1.230) and Turing, A. M. (1938). "On Computable Numbers, with an Application to the Entscheidungsproblem: A correction". *Proceedings of the London Mathematical Society*. 2. Vol. 43 (published 1937). pp. 544–546. doi:10.1112/plms/s2-43.6.544 (https://doi.org/10.1112%2Fplms%2Fs2-43.6.544). (See also: Davis 1965:115ff.)
- Alan Mathison Turing (December 1937). "Computability and λ-Definability" (https://web.archive.org/web/20200809215242/http://pdfs.semanticscholar.org/ee8c/779e7823814a5f1746d883ca77b26671b617.pdf) (PDF). *Journal of Symbolic Logic*. **2** (4): 153–163. doi:10.2307/2268280 (https://doi.org/10.2307%2F2268280). JSTOR 2268280 (https://www.jstor.org/stable/2268280). S2CID 2317046 (https://api.semanticscholar.org/CorpusID:2317046). Archived from the original (http://pdfs.semanticscholar.org/ee8c/779e7823814a5f1746d883ca77b26671b617.pdf) (PDF) on 2020-08-09.

External links

- "The Church–Turing Thesis" (https://plato.stanford.edu/entries/church-turing/) entry by B. Jack Copeland in the *Stanford Encyclopedia of Philosophy*.
- "Computation in Physical Systems" (https://plato.stanford.edu/entries/computation-physicalsystems/) entry by Gualtiero Piccinini in the *Stanford Encyclopedia of Philosophy*—a comprehensive philosophical treatment of relevant issues.
- Kaznatcheev, Artem (2014-09-11). "Transcendental idealism and Post's variant of the Church-Turing thesis" (https://egtheory.wordpress.com/2014/09/11/transcendental-idealism-and-posts-variant-of-the-church-turing-thesis/). *Journal of Symbolic Logic*. **1** (3): 103–105.
- A special issue (https://projecteuclid.org/euclid.ndjfl/1093637642) (Vol. 28, No. 4, 1987) of the *Notre Dame Journal of Formal Logic* was devoted to the Church–Turing thesis.